VŠB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

Biomedical Image Analysis using Deep Neural Networks

Použití hlubokých neuronových sítí pro analýzu biomedicínských obrazů

2018 Matúš Hromuľák

VŠB - Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

Bachelor Thesis Assignment

Student: **Matúš Hromuľák**

Study Programme: B2647 Information and Communication Technology

Study Branch: 2612R025 Computer Science and Technology

Title: Biomedical Image Analysis using Deep Neural Networks
Použití hlubokých neuronových sítí pro analýzu biomedicínských obrazů

The thesis language: English

Description:

Several Deep Learning techniques have been successfully applied for computer vision, natural language processing, image processing, time-series analysis. In this Bachelor project the student will develop and analyze Deep Neural Networks (NNs) in problems of image classification. In particular, the student will explore the potential of NNs of the family of Convolutioanl NNs, wich are a popular family of deep learning models. The students will develop the algorithms and apply them in real-world images related to mental health problems, such as Magnetic Resonance Image (MRI) and/or Functional MRI (FMRI). The algorithms will solve a binary classification of images. The thesis contains the following steps:

1. Theoretical revision of Deep Learning techniques and current state in the field of image classification.
2. Theoretical study of image processing.
3. Implementation of Deep Neural Networks.
4. Evaluation phase, testing and analysis.

References:

[1] Juergen Schmidhuber, Deep Learning in Neural Networks: An Overview, Neural Networks, Vol. 61, pages 85-117, 2015. Doi: 10.1016/j.neunet.2014.09.003, available at: https://arxiv.org/abs/1404.7828
[2] Yoshua Bengio, Deep Learning of Representations: Looking Forward, Department of Computer Science and Operations Research, University of Montreal, Canada, 2013. Available at: http://goo.gl/OK0WV9.
[3] Hastie, T., Tibshirani, R., Friedman, J., The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Second Edition, Springer, February 2009.
[4] Deep Learning information: http://deep learning.net.
[5] S. O. Haykin, Neural Networks and Learning Machines, Prentice Hall, third edition, 2008.

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: **Sebastian Basterrech Tiscordio, PhD.**

Date of issue: 01.09.2016

Date of submission: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
Head of Department

prof. Ing. Pavel Brandštetter, CSc.
Dean

I hereby declare that this bachelor's thesis was written by myself. I have quoted all the references I have drawn upon.

Ostrava, 30 April 2018 .....

I would like to thank my family, for the continuous support and sacrifices without which this thesis would never be finished. I would also like to thank my bachelor thesis supervisor, Sebastian Basterrech Tiscordio, Ph.D., for his professional guidance and consultation during the making of this thesis.

Abstrakt

Využívanie strojového učenia je v dnešných dňoch veľmi rozšírené a stále nové využitia sú postupne objavované. Jedno z prominentných paradigmat v strojovom videní sú konvolučné neurónové siete (CNN). Účelom tejto práce je priblížiť témy strojového učenia, konvolučných neurónových sietí a otestovať a vyhodnotiť experimentálne architektúry konvolučných neurónových sietí na obrazoch funkčnej magnetickej rezonancie (fMRI).

Séria rôznych viac-vrstvových CNN architektúr s alternujúcimi hyper-parametrami bola testovaná na dvoch všeobecne známych vzorových úlohách z oblasti klasifikácie obrazov: MNIST, CIFAR-10. Schopnosti modelov bol vyhodnotené na skutočnných fMRI obrazoch. Model siete bol znovu vytvorený s každým testom, obmieňajúc možné konfigurácie.

Navrhnuté modely vykazovali relatívne dobré výsledky na vzorových úlohách, ale neboli schopné prekonať súčasný stav vedy v klasifikácií obrazov mozgu. K získaniu možných lepších výsledkov, by bolo nutné ich rozšíriť, aby boli schopné absorbovať a rozlišovať medzi väčším množstvom atribútov. Takisto boli zistené limity použitého technického vybavenia a obmedzenia z nich vyplývajúce. Vychádzajúc z emprických výsledkov je možné vyhodnotiť CNN ako vhodný nástroj pre nachádzanie vzorov v obrazových dátach.

Klíčová slova: Hlboké Neurónové Siete, Konvolučné Neurónové Siete, Klasifikácia Obrazov, Analýza Biomedicínskych Obrazov, MNIST, CIFAR-10, fMRI

Abstract

The use of machine learning is very prevalent now-days and more new applications are continuously discovered. One of the prominent paradigms in computer vision are Convolutional Neural Networks (CNN). The purpose of this thesis is to introduce the topics of machine learning, convolutional neural networks and to test and evaluate experimental deep neural network architectures on functional Magnetic Resonance Images (fMRI).

A series of various multi-layer CNN architectures with alternating hyper-parameters was tested against two well-known benchmark problems in the area of image classification: MNIST, CIFAR-10. The models' capacity was also evaluated against a real-world dataset of fMRI images. The network's model was rebuilt with each test run, rotating between the possible configurations.

The proposed models, while performing relatively well on benchmark problems, were not able to surpass the current state of the art in brain image classification. To achieve possibly better results, they would need to be expanded to allow a broader set of features to be absorbed and classified. Also the limitations of the used hardware and the resulting impact were established. Based on the empirical results, it can be concluded that CNN are a viable tool for image pattern recognition.

Key Words: Deep Neural Network, Convolutional Neural Network, Image Classification, Biomedical Image Analysis, MNIST, CIFAR-10, fMRI

Contents

List of symbols and abbreviations

ABIDE	–	Autism Brain Imaging Data Exchange
Adam	–	Adaptive Moment Estimation
AI	–	Artificial Intelligence
ANN	–	Artificial Neural Network
API	–	Application Programming Interface
CNN	–	Convolutional Neural Network
CIFAR	–	Canadian Institute for Advance Research
cuDNN	–	NVIDIA CUDA Deep Neural Network library
DDSM	–	Digital Database for Screening Mammography
DL	–	Deep Learning
DNN	–	Deep Neural Network
fMRI	–	Functional Magnetic Resonance Imaging
FNN	–	Feed-forward Neural Networks
GPU	–	Graphics Processing Unit
MNIST	–	Modified National Institute of Standards and Technology
MRI	–	Magnetic Resonance Imaging
NN	–	Neural Network
ReLu	–	Rectified Linear Unit
SGD	–	Stochastic Gradient Descent
tanh	–	Hyperbolic Tangent

List of Figures

List of Tables

Listings

1 Introduction

Due to the economic need for more automation, less cost, more reliability and less risk, the field of Artificial Intelligence (AI) is experiencing large investments in many companies worldwide [1]. The economic factors are only one of the motivators for the development of computer aided decision making. The field of medicine offers a vast amount of challenges where the limitation is both the human capability and personnel capacity. The impact of under-staffing and miss-diagnosis is much more severe when human life and health is at stake.

The development of imaging technology in medicine has steadily progressed both in variety (computer tomography, magnetic resonance imaging) and quality. But for a long time, at the end of the day, a specialist had to sit down and analyse the results. That is a tedious and error-prone process. But thanks to the fact that **Machine Learning** (ML), a subsection of AI, does not necessarily require domain expertise to be successful, we can observe a high adoption rate rise in research done in the medical area [2]. A family of ML called **deep learning** (DL) has been especially successful when applied to visual recognition in specific domains [3].

While the base for **neural networks** has been laid as early as 1958 [4], they haven't seen real-world application until 1989 [5] due to the limitations of available computer computing capabilities. **Convolutional Neural Networks** (CNN) excel in pattern recognition in image data and are used to solve difficult task in this field [6]. Research to apply CNNs in medicine is tackling, for example, cell classification for lunger cancer [7], thoraco-abdominal lymph node detection [8] or brain tumour segmentation [9]

Medical equipment provides a large base of source data ripe with potential applications for analysis. **Functional Magnetic Resonance Imaging** (fMRI), the data source used for this thesis, is a widely adopted non-invasive brain imaging technology used in medicine and research. This is achieved by observing the change in blood flow as brain cells fire off and consume energy [10]. The result is a series of three-dimensional brain images where, if colour coded, sections of the brain can be observer to "light up" as the brain performs certain tasks.

There are already examples of deployment of AI in medicine helping in daily life, although just preliminary [11]. There are expectations of adoption of at least some form of AI within the next two to five years. This come of course with it's own set of **challenges**: general mistrust in IT, the solutions usability, scalability, maturity, stakeholder commitment, state- and international-level regulations, to name a few [12, 2].

This thesis explores the problem of using convolutional neural networks to analyse graphical data, learning their features and making meaningful predictions based on the learning process. The developed algorithm will be tested against the MNIST and CIFAR-10 benchmark problems. Then it will be applied on actual biomedical data from the StarPlus fMRI study where

its ability to differentiate between mental states will be evaluated. The **objective** of this thesis is to evaluate the proposed CNN models, their capacity, effectiveness to learn and to infer their further development.
Chapter 2 discusses the current state of art of machine learning. Chapter 3 introduces convolutional neural networks and the algorithm produced as part of this thesis. Chapter 4 describes the used technologies and implementation. Chapter 5 provides the results gained from analysing benchmark machine learning problems. Chapter 6 presents the outcome of applying the created neural network on biomedical data.

2 State of art

Machine learning is a subclass of AI. One of the applications is **computer vision** [13], where an algorithm is thought how to recognize patterns in image data to come up with a specific conclusion. The result being, for example, whether a specific sample does show signs of illness. The rise of interest in AI applications in medicine is reflected in the amount of research that has been performed over the last years. The graph in figure 1 illustrates this based on search hist for keywords "machine learning" and "Deep Learning" (DL) on the PubMed website [14].

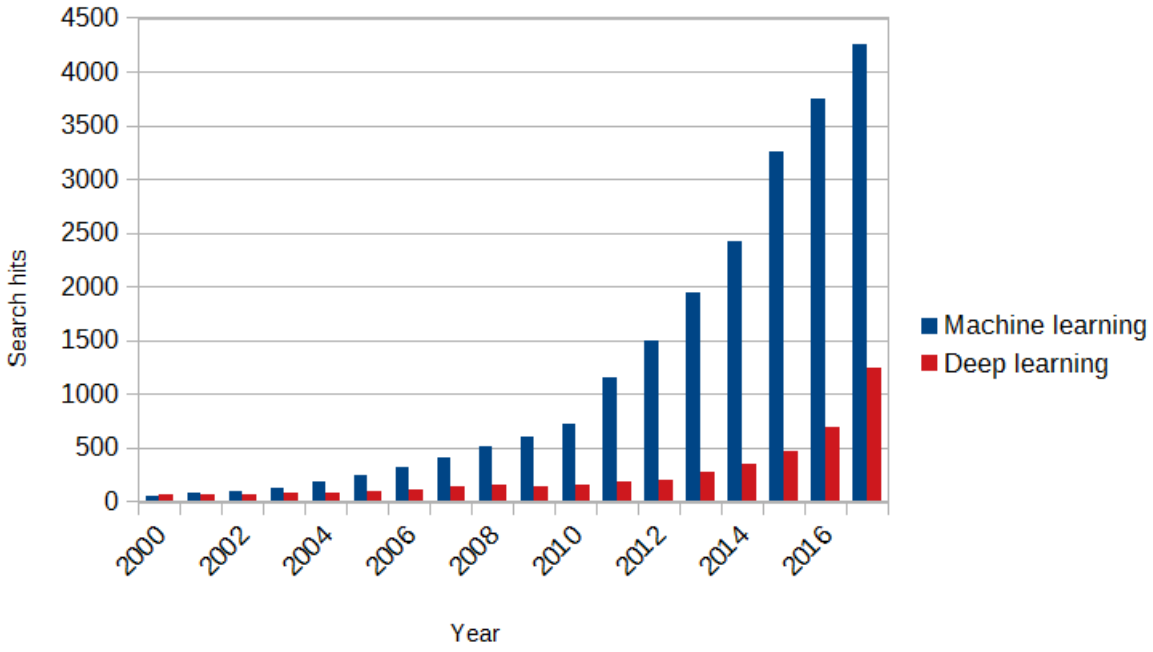


Figure 1: Number of search hits for "machine learning" and "deep learning" on PubMed.

2.1 Machine learning problems

Machine learning can be broken down to two approaches, supervised and unsupervised learning [15]. For **unsupervised learning**, the data is left without additional labelling and it is up to the algorithm to makes sense of it. This thesis focuses on **supervised learning**. During training in this case, the algorithm is provided both data and the expected result, thus able to adjust itself to more accurately predict the outcome the next turn around [16].

How well the algorithm is able to learn during the training process is highly dependent on the **amount, variety and quality of available data** [17]. **Deep learning** refers to machine learning problem where several non-linear transformations are applied in order to obtain mapping between the input and the output information [18, 19].

2.1.1 Classification

Classification in machine learning is an application of pattern recognition where the problem of identifying the category type of input data is decided. A **classifier** may, for example, sort incoming images on whether the picture contains a specific object. It would be able to tell apart images that contain chairs, lamps etc. For example in the case that the input variable belongs to the real space, and the task is to predict a natural number in $\{1, 2, \ldots, k\}$ given a real number, then the classifier algorithm produces a function:

$$f : \mathbb{R} \to \{1, \ldots, k\}, \text{ for } y = f(x). \tag{1}$$

The model assigns an input $x \in \mathbb{R}$, to a category $y \in \{1, 2, \ldots, k\}$ [15].
The model $f(\cdot)$ can be any type of parametric function, in the case of this thesis the used classifier algorithm is a specific type of neural networks named **deep neural networks**

2.1.2 Training/testing cross-validation

Before training, the available data are typically split into two parts: **training** and **test** data (80:20 ration for example). The algorithm attempts to learn the useful features from the training data and is tested on the test data.

An algorithm that learned to recognize certain patterns in the training data may perform badly when facing previously unknown data. This is called **over-fitting**, it is unable to recognize new patterns due to modelling even minor features of the training data. The training metrics are validated against the test dataset. There are also other approaches to counter over-fitting, e.g. a drop out rate.

2.1.3 Metrics

To evaluate how well the algorithm performs on the given task various metrics are available. **Accuracy** expresses the percentage of correct classifications performed against training data. **Validation accuracy** does the same, but against validation data.

Loss and **validation loss** follow the same relationship. Loss is calculated through a **loss function**. It transforms one or more variables into a single overall measure of loss, or cost, that will be taken in case of any performed action [20]. The more the predictions of the model are wrong, the higher the loss. As the models makes better and better predictions, the loss decreases.

Cross-entropy loss is used for classifications problems. Let's denote θ as the complete set of weights , f as the estimate for the true distribution y over x, $x_1, \ldots, x_N$ are the input patterns

and $y_{1,1}, \ldots, y_{N,K}$ are the target matrix variables, the function is [15]:

$$J(\theta) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \log f_k(x_i).$$

2.1.4 Bias/variance dilemma

Estimation errors can be split into two components, bias and variance.

Bias are the assumptions made in the learning algorithm, and incorrect model leads to high bias. Under-fitting is a result of high bias, the algorithm fails to learn the relevant features [16].

Variance is the sensitivity of the algorithm to training data details. A high variance may model noise in the data and cause to over-fit. However, some variance is expected so the model is able to learn. As the model's complexity increases, so does variance and bias tends to decrease. A reduction of the model's complexity has the opposite effect [15, 21].

Irreducible error, also "*noise*", cannot be reduced via algorithms as it is usually inherent to the source data [16].

2.1.5 The curse of dimensionality

In most practical applications, input data is pre-processed into a format which is hoped to be easier processable and the problem easier to solve. For image data, the images may be scaled, translated and cropped to fit a single specific format. This process is also called **feature extraction** [20].

Through feature extraction we can reduce or increase the amount of data the learning algorithm can analyse to find the distinguishing features of the solution to a problem. An increase of features can lead to an *increase* of performance. In real application, there is however a certain point where adding new features can *reduce* the performance [21].

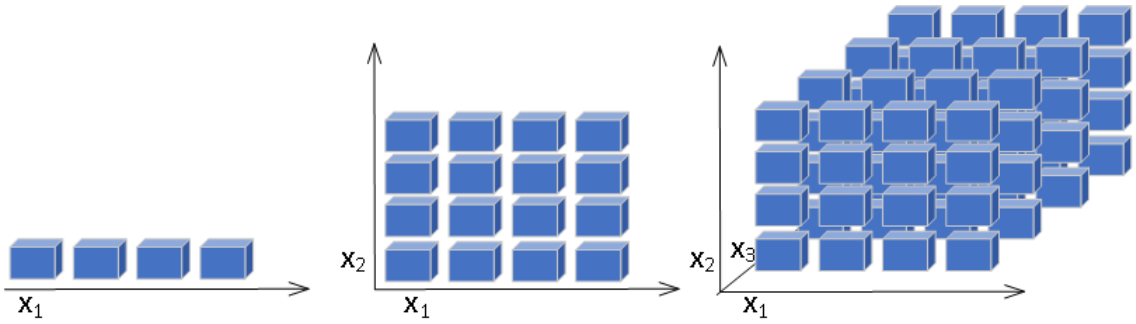


Figure 2: The curse of dimensionality.

Imagine an n dimensional space and a function f that maps that space to a single variable y. If we want to map the whole available space, run the function for all possible combinations, with the increase of the number of dimensions, the number of computations for the mapping

function increases exponentially. This is described as the **curse of dimensionality** [22]. A simple illustration is presented in figure 2, where the mapping is represented by the blue

2.2 Neural network

Neural Networks (NN) are a machine learning paradigm quasi-inspired by the biology of the nervous systems.

"*A neural network can be described as a directed graph whose nodes correspond to neurons and edges correspond to links between them. Each neuron receives as input a weighted sum of the outputs of the neurons connected to its incoming edges.*" [23]

2.2.1 Neuron

A **neuron** is a single node in the NN and bares the weight of the computation within the model. They are modelled after the biological neuron (Figure 3) contained in the nervous systems.

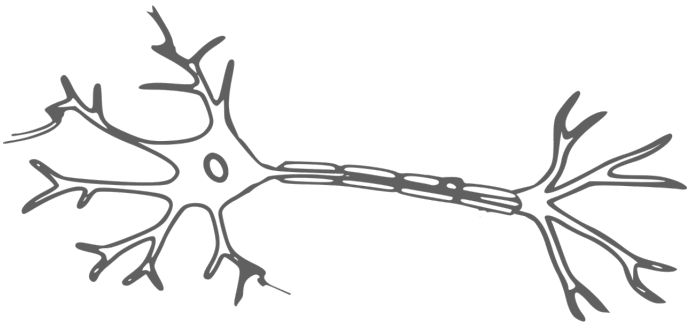


Figure 3: Biological neuron [24].

The NN neuron can be expressed as: where

- $\boldsymbol{x_i}$ is the input from the $\boldsymbol{i}$th neuron
- $\boldsymbol{w_i}$ is the weight for the input from the $\boldsymbol{i}$th neuron
- $\boldsymbol{b}$ is the bias
- $\boldsymbol{f}$ is the activation function
- $\boldsymbol{y}$ is the output [25].

They were first proposed by Frank Rosenblatt as **perceptrons** [4]. The neurons are linked together to perform complex computations. A single neuron takes in the input from all forward-connected neurons, sums it, applies the activation function and provides the resulting output on the other end, illustrated by figure 4.

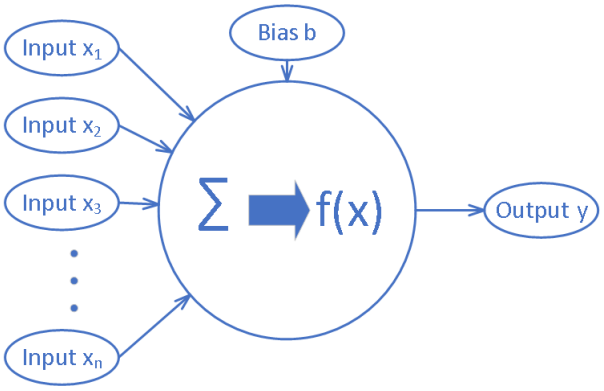


Figure 4: Neural network neuron.

2.2.2 Network topology

Neurons are connected into **multi-layered networks**, compromised of an input layer, an output layer and one ore more hidden layer between them. The **hidden** layers are called hidden due to the fact that they are not normally visible. Compared to the **input** layer which represent the data intake, and the **output** layer which provides the result from the model. This relationship is shown in figure 5. A neural network that does not contain cycles is called a **Feed-forward Neural Networks** (FNN).

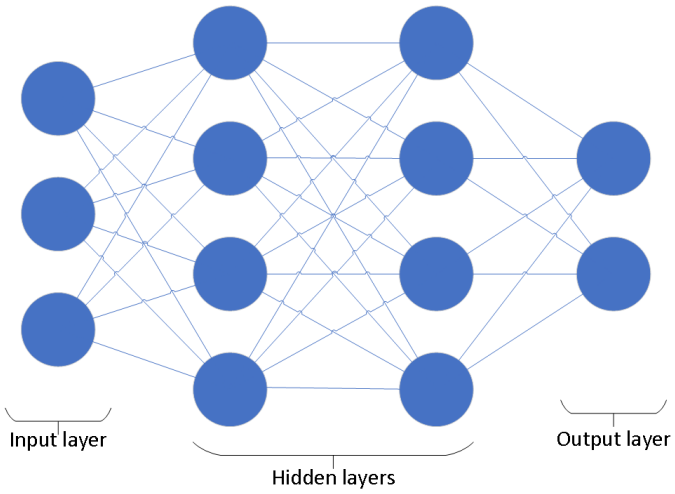


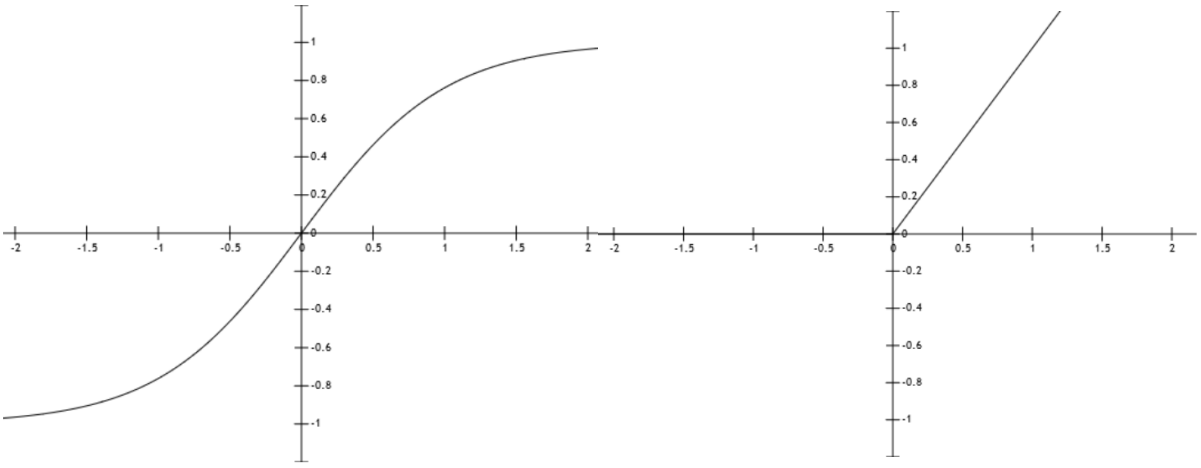
Figure 5: Multi-layer neural network.

Besides FNN, there exist other Artificial Neural Network (ANN) topologies such as recurrent ANNs, Hopfield ANNs, Elman and Jordan ANNs, bi-directional ANNs, self-organizing maps and stochastic ANNs [25].

2.2.3 Activation functions

The input of a neuron, the previous results x multiplied by their corresponding weight w and the added offset bias b, are summed and processed by the non-linear **activation function**, also called *threshold function*. The function computes the result for that neuron [26].

An example of activation function used in this thesis would be the **Rectified Linear Unit** (ReLu) and the **Hyperbolic Tangent function** (tanh). Tanh, $f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, transforms real number values into the range $\langle -1, 1 \rangle$ (Fig. 6a). ReLu computes the function $f(x) = max(0, x)$, cutting off all negative values (Fig. 6b).



(a) Tanh activation function

(b) ReLu activation function

Figure 6: Plots of activation functions.

2.2.4 Optimization functions

The FNN takes in an input and computes and output. The information flows through the network from the first layer to the last, this is described as **forward propagation**. An algorithm such as **Stochastic Gradient Descent** (SGD) is performing the learning by doing computations using a gradient calculated and provided by **back-propagation**. The forward propagation provides a cost, or loss, at the end of the initial computations and the back-propagations pushes the information from this cost backwards from the last layer [16].

"*Gradient descent is a way to minimize an objective function $J(\theta)$ parametrized by a model's parameters $\theta \in \mathbb{R}^d$ by updating the parameters in the opposite direction of the gradient of the objective function $\nabla_\theta J(\theta)$ w.r.t. to the parameters. The learning rate η determines the size of the steps we take to reach a (local) minimum.*" [27]

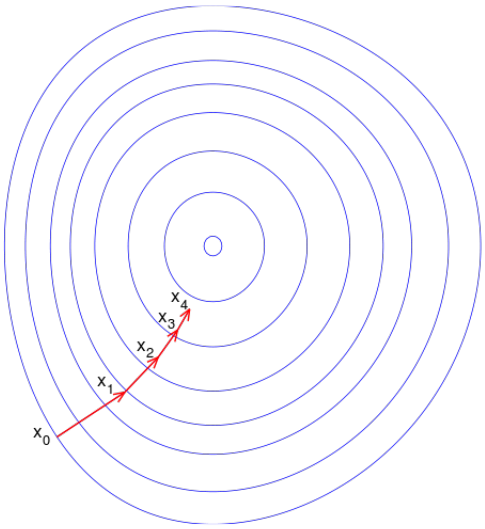


Figure 7: Gradient descent. [28]

SGD updates parameters one after another for all training examples $x^{(i)}$ and labels $y^{(i)}$: $\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$. SGD has a high level of fluctuation, which enables it to jump curves to explore other local minima. With decreasing learning rate, SGD shows confident convergence behaviour towards a local or global minimum [27]. This thesis utilizes a gradient descent variant with an adaptive learning rate.

Adaptive Moment Estimation (Adam) is an algorithm for gradient optimization which computes the learning rates for each parameter. The Adam update rule is $\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t$, where $\epsilon = 10^{-8}$, $\hat{v}_t$ and $\hat{m}_t$ are bias-corrected first and second moment estimate, which are based of initial first and second moment estimates (v_t and m_t) [29].

2.2.5 Back-propagation

Regularization desribes a set of strategies for reducing testing error, including at the expense of training error [16]. Regularization has not been utilized in this thesis, but is listed for completion.

Let's propose a neural network with a depth l, a weight matrix $w^{(i)}, i \in \{1, \ldots, l\}$, bias parameters $b^{(i)}, i \in \{1, \ldots, l\}$, the input x, the target output y, f an arbitrary function $f : \mathbb{R} \to \mathbb{R}$ and the regularization term λ. The output $\hat{y}$ for the target y define the loss $L(\hat{y}, y)$. We can obtain the **total cost** $\boldsymbol{J}$ with the addition of the loss to the regularizer $\boldsymbol{\Omega(\omega)}$, $\boldsymbol{\omega}$ being the weights and biases [16].

```
#input
h^(0) = x
#inter-neuron calculation
for k in range(1, l):
    a^(k) = b^(k) + w^(k) * h^(k-1)
    h^(k) = f(a^(k))
#output
ŷ = h^(l)
#calculate cost
J = L(ŷ, y) + λ * Ω(ω)
```

Listing 1: A pseudocode for forward propagation with a single input.

Back-propagation yields the gradients for each layer k on the activation $a^{(k)}$. This computations follows the network from the last to the first layer. From this gradient, we can compute the gradients on weights and biases [16].

```
#output layer
g ← ∇_ŷ J = ∇_ŷ L(ŷ, y)
for k in range(l, 1):
    #convert the gradient on the layer's output into a gradient into the pre-
        nonlinearity activation
    g ← ∇_{a^(k)} J = g ⊙ f'(a^(k))
    #compute gradients on weights and biases
    ∇_{b^(k)} J = g + λ∇_{b^(k)} Ω(ω)
    ∇_{w^(k)} J = g h^(k-1)T + λ∇_{w^(k)} Ω(ω)
    #propagate the gradients to the next level
    g ← ∇_{h^(k-1)} J = W^(k)T g
```

Listing 2: A pseudocode for back-propagation.

2.3 Machine learning and neural networks application in biomedical analysis

The steady increase of computational power and data availability, especially image data, aids the rise of supervised machine learning. Such models are used to **diagnose**, **predict future diagnosis** and **drug discovery** in the medical field [30]. This is further supported by open access to labelled medical data sets. The **Autism Brain Imaging Data Exchange** (ABIDE) database provides 1112 datasets of fMRI and MRI data from individuals with autism spectrum disorder and standard controls. The **Digital Database for Screening Mammography** (DDSM) contains 2500 studies which can be used for mammographic image analysis to aid in

breast cancer diagnosis. However, application in day-to-day life has many issues. According to [31], this can be summed into three main points:

- **Varying imaging protocols** - training data often comes from a specific use case. That incorporates the imaging protocol, scanner model, patient sample, etc. This can be countered by an extensive and heterogenic dataset or learning domain adaptation techniques.
- **Weak labels** - the problem of the lack of quality training data is further deepened by the lack of labelled data. It is required for a domain expert to correctly label and process available data solely for the purpose of further image analysis.
- **Interpretation and evaluation** - machine learning models are often regarded as a "black box". A correct diagnosis might not have been drawn from the actual signs of a disease, but might have been based on correlating factors.

The worry of not knowing how an AI actually reasons is prevailing and will be one of the hurdles that will have to be successfully tackled, or market pressure will be great enough for a risky "leap of faith" [32].

3 Convolutional neural networks

Convolutional neural networks are biologically inspired NNs, where the design is based on how the visual cortex processes images. The first computational model based on this idea, nicknamed "**neocognitron**", was proposed by Kunihiko Fukushima [33]. This was later built upon by Yann LeCun et al. [5] where a convolutional network was applied to recognize handwritten digits.

3.1 Convolutional layer

Convolutional networks are named after the mathematical operation **convolution**. Convolution is an operation on two functions, producing a third function. This is expressed as [16]:

$$\boldsymbol{s(t) = (x * w)(t).}$$

In this example, the input would be the function $\boldsymbol{x}$, the kernel the function $\boldsymbol{w}$. The output is the **feature map**. Applying a two-dimensional kernel $\boldsymbol{K}$ on a two-dimensional image $\boldsymbol{I}$, where $\boldsymbol{m, n}$ and $\boldsymbol{i, j}$ are the dimensions [16]:

$$\boldsymbol{S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n).}$$

The process can be visualized as a window, representing the kernel, sliding over an image with a specific **stride**, size of "steps", and providing the output as a new layer, the **convolutional layer**. This operation is shown in figure 8.

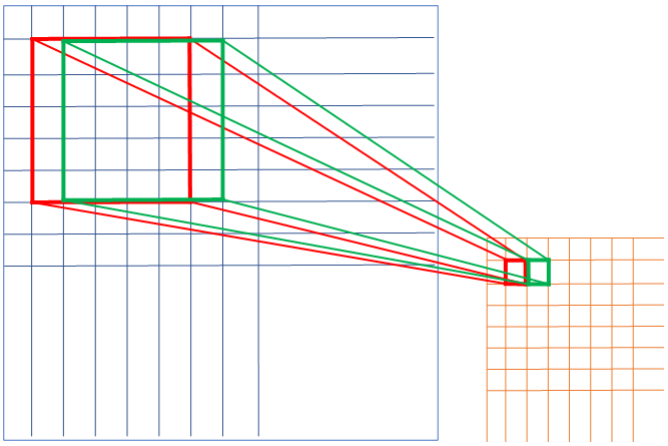


Figure 8: Convolution operation on 2D image data with a five by five kernel and a stride of one.

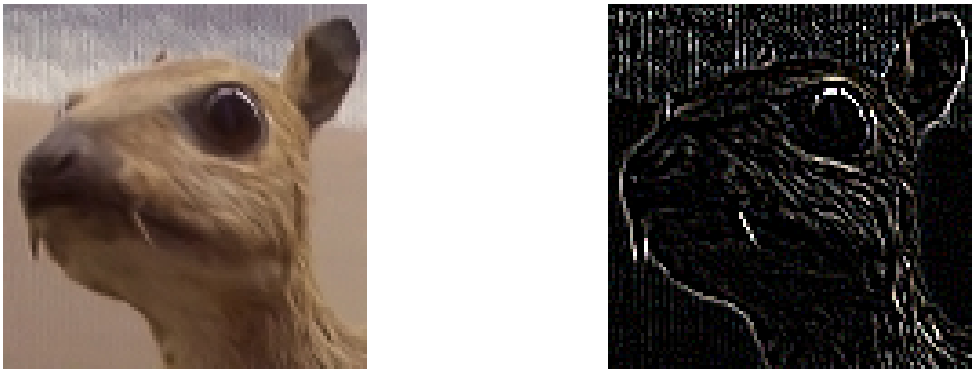
The **kernel** is a matrix of the same dimenaion and typically smaller then the input image [16]. The content of the matrix defines the resulting after the convolution operation. For example, a

3x3 edge detecting kernel would look like this:

$$\begin{bmatrix} \mathbf{-1} & \mathbf{-1} & \mathbf{-1} \\ \mathbf{-1} & \mathbf{8} & \mathbf{-1} \\ \mathbf{-1} & \mathbf{-1} & \mathbf{-1} \end{bmatrix},$$

and the application on an image can be seen in figure 9.

If the kernel is capable of detecting meaningful features (e.g. horizontal and vertical edges) it can reduce the number of parameters that need to be analysed. This is called **sparse connectivity** [16].

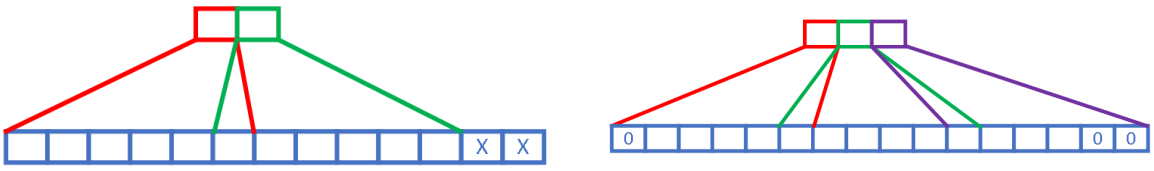


(a) Original image (b) After edge detection

Figure 9: Image processing with an edge detection filter kernel [34, 35].

This approach **extracts local features** from small subsections of the image. Suppose the kernel would be the size of five by five, that would yield twenty-five adjustable weights. These **weights** are **shared** for the whole input. Therefore, we can think of the kernel as a **feature detector** [20] as it will detect the same features from the whole image.

Depending on the size of the kernel and the applied stride, it may happen that a part of it input may be left unprocessed. This is addressed by **padding** [36]. On a one-dimensional example, let's have a thirteen point line, a kernel of size six and stride five. In case of **valid** padding (Fig. 10a), the leftover data point would be ignored. In case of **same** padding (Fig. 10b), the data sample would be extended by adding zeros so the kernel fits the sample. [36]



(a) Valid padding (b) Same padding

Figure 10: Padding example.

Another CNN specific property is **equivariance**. This means, that transformations applied to the input and run through convolution have the same effect as if we processed the original image and applied the transformations on the result. It is not true for all kind of transformations, changes in scale and rotations are excluded [16].

To compute the output of a single neuron in a convolutional layer:

$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_n-1} x_{i',j',k'} \cdot w_{u,v,k',k} \quad \text{with} \quad \begin{cases} i' = i \times s_h + u \\ j' = j \times s_w + v \end{cases},$$

where $z_{i,j,k}$ is the neuron output in row i, column j and feature map k, f_h and f_w are the height and width of the receptive field, f_n is the number of feature maps in the previous layer, s_h and s_w are the strides, $x_{i',j',k'}$, is the output of the neuron in the previous layer, b_k is the bias, $w_{u,v,k',k}$ is the weight between neurons on the k and k' feature maps, u, v being the coordinates [36].

3.2 Pooling and fully-connected layers

The **pooling layer**, also described as sampling layer, is commonly inserted in between or after convolutional layers (approaches where CNN are formed only through convolutional layers are also used [37]). They are used to reduce the size of the representation, parameter reduction and as a over-fitting measure [38].

For example, a **max pooling** layer with a two by two filter size and a stride of two would take in a block of four cells and reduce them to one, picking the maximum value of the input cells as the output value for the new cell. An example of this operation is shown in figure 11.

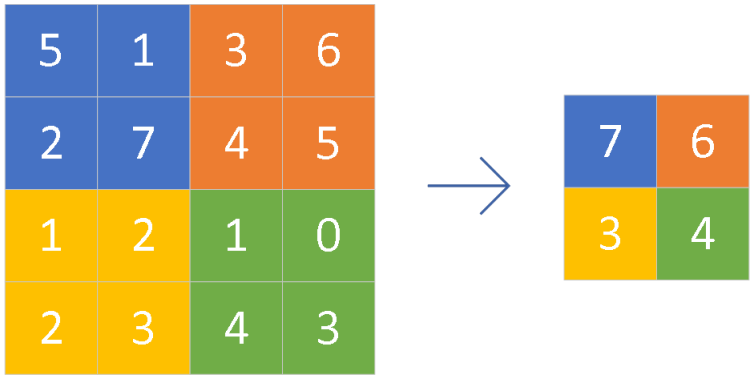


Figure 11: Pooling layer example, max pooling.

Fully-connected layers, as the name suggest, have a full set of connections to all neurons in the previous layer. In the Keras Application Programming Interface (API), defined in chapter 4, these are called "*dense*" layers.

3.3 Model definition

In general, the **architecture of a CNN** would consist of alternating convolutional and pooling layers, followed by a fully-connected layer. Many architectures have been developed in the past, to name a few examples:

- **LeNet** [39] - a two-layered CNN, developed by Yann LeCun et al., used to read in digits from zip codes
- **AlexNet** [40] - an eight-layered network with five convolutional layers, developed by a team from the University of Toronto, entered in the ILSVRC 2010 contest
- **GoogLeNet** [41] - also known as "Inception", a twenty-seven-layered network, developed by a team from Google, entered in the ILSVRC 2014 contest
- **VGGNet** [42] - a nineteen layered network, developed by a team from the University of Oxford, entered in the ILSVRC 2014 contest
- **ResNet** [43] - a one hundred fifty-two layered network, developed by a team from Microsoft, entered in the ILSVRC 2015 contest

While some literature exists on various techniques how to build and optimize NNs [44], or empirical results on which approaches seem to work well [45], there is no single heuristic or design principle to follow. But some guidance can be obtained from the sum of research papers in this field. A NN design, according to [46], should follow these **principles**:

1. *Architectural Structure follows the Application*
2. *Proliferate Paths*
3. *Strive for Simplicity*
4. *Increase Symmetry*
5. *Pyramid Shape*
6. *Over-train*
7. *Cover the Problem Space*
8. *Incremental Feature Construction*
9. *Normalize Layers Inputs*
10. *Input Transition*
11. *Available Resources Guide Layer Widths*

12. Summation Joining

13. Down-sampling Transition

14. Maxout for Competition

The experimental CNN architectures used in this thesis are discussed in chapters four and five.

4 Implementation

For this thesis, four separate programs were created. The programs `MNIST_CNN.py`, `CIFAR10_CNN.py` and `fMRI_CNN.py` implement the CNN for the MNIST, CIFAR-10 and StarPlus fMRI datasets. An additional support program, `fMRI_matfile.py`, was created to extract and remodel the data sourced from the StarPlus dataset into a three-dimensional array that can be loaded into a convolutional layer. The **source code** and the **manual** are available on the attached CD.

4.1 Used technologies

The following technologies were used in the development and testing of the models discussed in this thesis:

- **Python** - the source code was written in Python, a general purpose, high-level, interpreted programming language
- **GPU** - the computations were performed on a Nvidia GTX950M graphics card
- **cuDNN** - a GPU library, provides routines for deep neural networks
- **TensorFlow** - open-source software library for machine learning
- **Keras** - high-level neural network API, using TensorFlow as computing back-end

CNNs can be very demanding when it comes to computational resources, largely due to the amount of data they need to process to be effective. The available **hardware** poses a great limitation on what is realistically possible.

4.2 Implementation description

The experimental testing of CNN models follows a simple code structure:

```
parse_arguments()
prepare_output_folders()
training_data, test_data, training_labels, test_labels = process_input()
prepare_hyperparameters()
for configuration in hyperparameter_configurations:
   model = create_model(configuration)
   model = train_model(model, training_data, test_data, training_labels,
      test_labels)
   evaluate_model(model, test_data, test_labels)
   provide_output_files()
   delete model
```

Listing 3: A simplified high level pseudo-code implementation.

A model is trained on the same training data for a specific number of **epochs**. The epoch defines one training cycle, at the end of which, the performance of the model is validated on both the training and the validation data. However, the validation data are not used during the training. Only the results from the test data are used for parameter updates. This allows us to see how well the model performs and extracts and learns features.

The CNN programs support real time **data augmentation**. In case of small training sample datasets, it is possible to create "additional" training data by performing various transformation (rotation, scaling, positional shift, colour shift) on the data. This feature was not utilized as the tested datasets provided sufficient samples.

The CNN programs provide a series of data outputs:

- **Hyper-parameter export** - the hyper-parameters utilized during training are listed
- **Training history** - a log of the training progress, the accuracy, loss, validation accuracy and validation loss are generated at the end of each epoch
- **Architecture export** - the model's architecture can be exported into a .json file, the architecture can so be reused
- **Model data** - the trained model can be exported (the weights and biases), such a model can be later loaded and used for further training or data analysis
- **Tensorboard logs** - Tensorboard is a suite of visualization tools to allow easier insight into the models

Figure 12 shows a visualization through Tensorboard of a CNN model with two 2D convolutional layers connected to a pooling layer (with MaxPooling function) and then two fully connected layers, the first of which has a dropout operation applied.

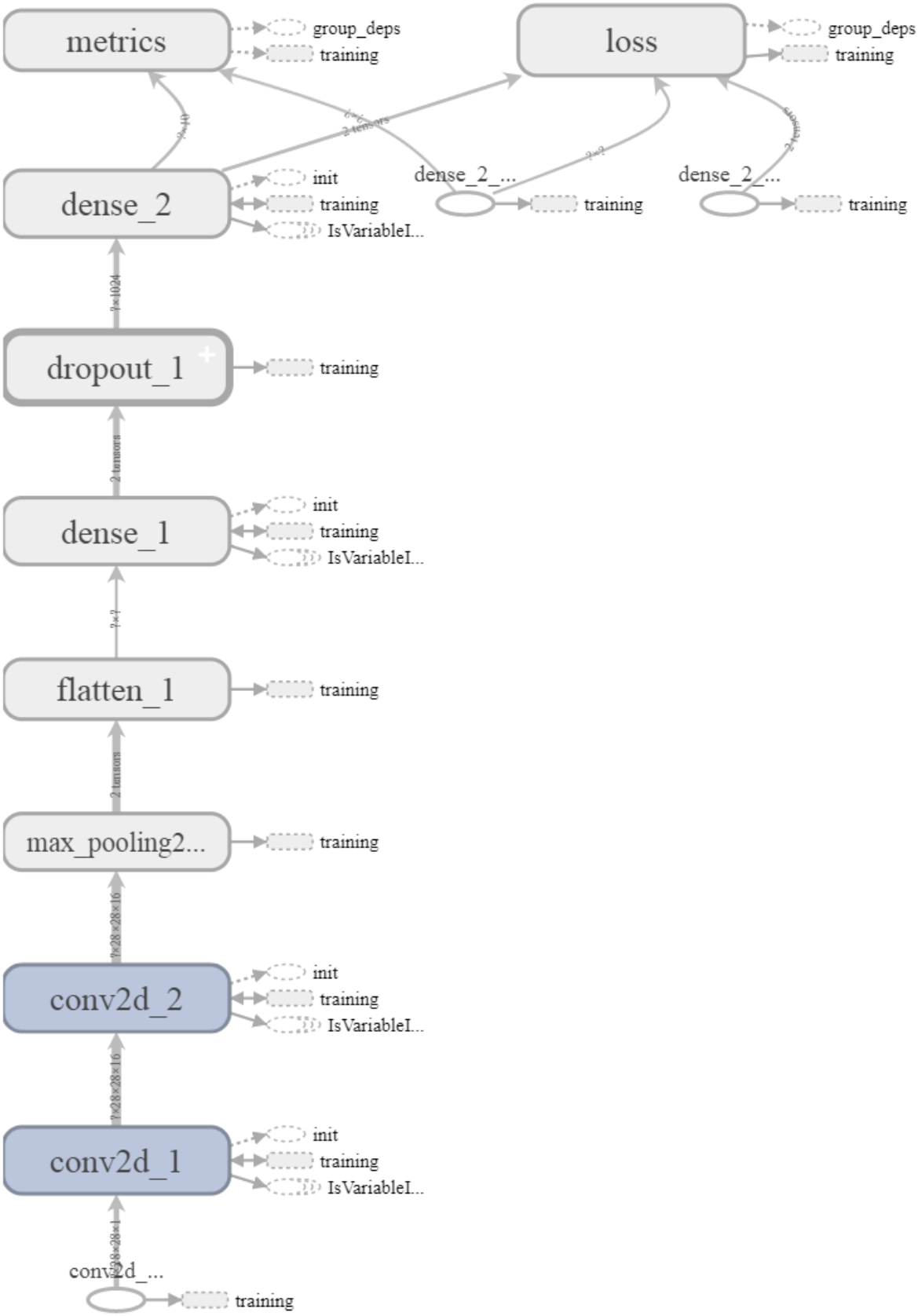


Figure 12: An example of a CNN visualized in Tensorboard.

5 Experimental results on well-known benchmark problems

This section examines the experimentation with multiple CNN architectures on well-known benchmark problems.

5.1 MNIST

The **MNIST** benchmark problem is based on identifying **hand-written digits**. The dataset consists of 60 000 training images and 10 000 testing images. It provides a good starting point for initial patter recognition learning stages. The MNIST dataset was built from NIST's Special Database 1 and Special Database 3. [47]

5.1.1 Data description

The MNIST dataset images are already pre-processed. They are black and white images of 28x28 pixel size, centralized by the computed pixel mass.

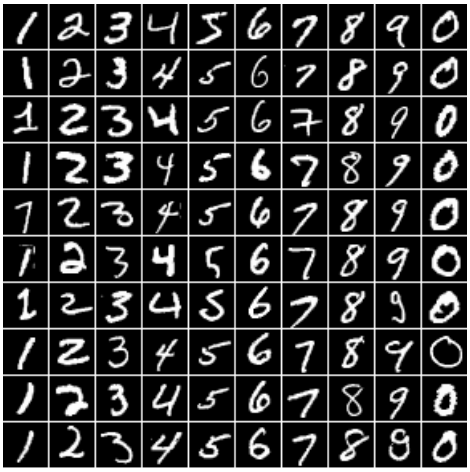


Figure 13: MNIST database of hand-written digits, a sample.

5.1.2 Methodology

Various combinations of CNN architectures have been tested on this set. The set is presented on a loop to an algorithm rebuilding the CNN before every test, changing one of the parameters, essentially "**brute-forcing**" the best available result. While there are many hyper-parameters available with a myriad of possible setting values, to explore the full landscape is an impossible task. The number of test cases rises exponentially which each additional parameter or value. A sensible set of hyper-parameters and values was selected for experimental evaluation based on prior domain knowledge (Tab 1) [38, 48].

To distinguish between the different test runs a simple **naming convention** for the output was devised. All changeable parameters provide the first letter of their value in a specific order.

Table 1: Used hyper-parameters for the MNIST test.

Hyper-parameter	Values
Activation function	relu, tanh
Batch size	50
Dropout rate	0.5
Number of epochs	7
Initial bias	0.1
Number of layers	2, 4, 6
Optimizer functions	SGD, Adam
Loss function	Categorical Crossentropy
Pooling functions	Max Pooling, Average Pooling
Number of neurons	16, 32, 64
Kernel size	3
Pool size	2
Padding	same

A test with the **T**anh activation, **4** convolutional layers, **S**GD optimizer, **A**verage pooling and **32** neurons would be encoded as **t4sa32**. The convolutional layers increase by two, with each subsequent layer doubling the number of neurons. So a model with six convolutional layers and 16 neurons would have 16 neurons in the first two layers, 32 in the third and fourth, 64 in the fifth and sixth convolutional layer.

The model's **architecture** is made from blocks of two convolutional layers with a pooling layer. These three layers repeat once or twice more for four and six layered configurations. The output is collected into a fully-connected layer of 1024 neurons. A drop-out rate of 50%, half of the inputs are randomly selected and discarded, is applied and then the result encoded into a ten-neuron fully-connected output layer.

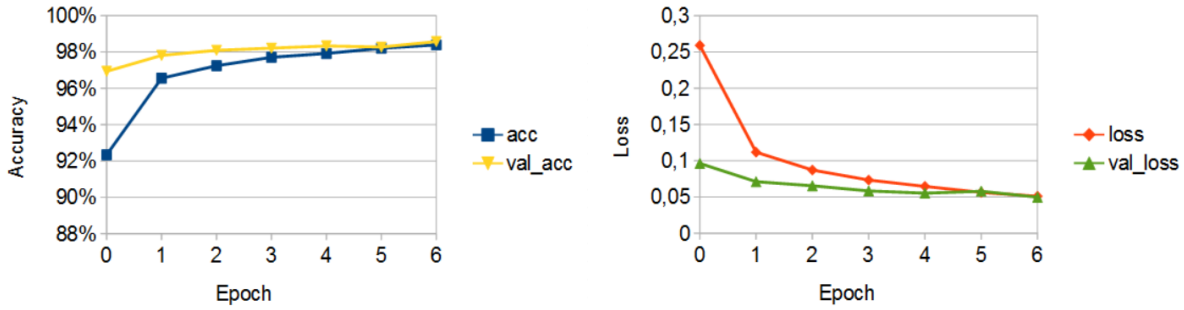
Table 2: Architecture of an r6aa16 CNN model (generated by the "–hyp" program parameter).

Layer (type)	Output Shape	Params #
conv2d_1 (Conv2D)	(None, 28, 28, 16)	160
conv2d_2 (Conv2D)	(None, 28, 28, 16)	2320
average_pooling2d_1	(None, 14, 14, 16)	0
conv2d_3 (Conv2D)	(None, 14, 14, 32)	4640
conv2d_4 (Conv2D)	(None, 14, 14, 32)	9248
average_pooling2d_2	(None, 7, 7, 32)	0
conv2d_5 (Conv2D)	(None, 7, 7, 64)	18496
conv2d_6 (Conv2D)	(None, 7, 7, 64)	36928
average_pooling2d_3	(None, 4, 4, 64)	0
flatten_1 (Flatten)	(None, 1024)	0
dense_1 (Dense)	(None, 1024)	1049600
dropout_1 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 10)	10250

The table 2 shows an example of the r6aa16 CNN architecture. It lists the layers as they are stacked after another, their shape and the count of trainable parameters. So the first line is a two-dimensional convolution layer, it takes in a 28x28 image, has 16 filters and 160 trainable parameters. The weight count is calculated as $(\boldsymbol{h} * \boldsymbol{w} * \boldsymbol{c} + \boldsymbol{b}) * \boldsymbol{f}$, where $\boldsymbol{h}$ is the kernel height, $\boldsymbol{w}$ the kernel width $\boldsymbol{c}$ the channel count, $\boldsymbol{b}$ the bias count and $\boldsymbol{f}$ the number of filters ($\boldsymbol{(3 * 3 * 1 + 1) * 16 = 160}$).

5.1.3 Results

The tables 3 and 4 present complete results from all run tests. They show the validation loss, validation accuracy and the time it took to train the model.
There are four metrics presented in the below tables, acc, val_acc, loss and val_loss meaning **accuracy**, **validation accuracy**, **loss** and **validation loss**. Accuracy and loss are calculated during the training against the training data. Validation accuracy and validation loss are calculated against validation data. The latter two metrics represent how well the model would fare against previously unknown data.
The table 5 and figure 14 presents the learning process for a CNN composed of: relu activation function, two convolutional layers, adam optimizer function, average pooling and sixteen neurons in the two convolutional layers (encoded as **r2aa16**).



(a) Accuracy

(b) Loss

Figure 14: Learning process for the MNIST CNN r2aa16 configuration.

As another sample (Tab. 6, Fig. 15), a CNN composed of: relu activation function, six convolutional layers, adam optimizer function, max pooling, thirty-two neurons in the first two, sixty-four in the second two and one hundred, twenty eight in the third two convolutional layers (encoded as **r6am32**).

Table 3: Complete results from MNIST experimental CNNs (ReLu activation).

Activ.	**Layers**	**Optimizer**	**Pooling**	**Neurons**	**Loss**	**Accuracy**	**Time (s)**
relu	2	SGD	Max	16	0.0647	0.9786	219
relu	2	SGD	Max	32	0.0719	0.9783	358
relu	2	SGD	Max	64	0.0703	0.9775	860
relu	2	SGD	Average	16	0.0715	0.9773	380
relu	2	SGD	Average	32	0.0709	0.9777	562
relu	2	SGD	Average	64	0.0722	0.9781	1 042
relu	2	Adam	Max	16	0.0372	0.9892	404
relu	2	Adam	Max	32	0.0326	0.9909	508
relu	2	Adam	Max	64	0.0355	0.9901	1 042
relu	2	Adam	Average	16	0.0497	0.9857	458
relu	2	Adam	Average	32	0.0341	0.9893	572
relu	2	Adam	Average	64	0.0352	0.9891	1 040
relu	4	SGD	Max	16	0.0320	0.9887	222
relu	4	SGD	Max	32	0.0340	0.9880	396
relu	4	SGD	Max	64	0.0346	0.9881	1 128
relu	4	SGD	Average	16	0.0471	0.9851	402
relu	4	SGD	Average	32	0.0455	0.9851	596
relu	4	SGD	Average	64	0.0456	0.9848	1 157
relu	4	Adam	Max	16	0.0223	0.9935	268
relu	4	Adam	Max	32	0.0226	0.9930	475
relu	4	Adam	Max	64	0.0333	0.9904	1 201
relu	4	Adam	Average	16	0.0228	0.9919	265
relu	4	Adam	Average	32	0.0218	**0.9936**	464
relu	4	Adam	Average	64	0.0846	0.9776	1 208
relu	6	SGD	Max	16	0.0352	0.9869	314
relu	6	SGD	Max	32	0.0296	0.9898	557
relu	6	SGD	Max	64	0.0308	0.9890	1 552
relu	6	SGD	Average	16	0.0543	0.9810	434
relu	6	SGD	Average	32	0.0571	0.9820	701
relu	6	SGD	Average	64	0.0532	0.9821	1 561
relu	6	Adam	Max	16	0.0333	0.9894	270
relu	6	Adam	Max	32	0.0238	0.9935	471
relu	6	Adam	Max	64	0.0349	0.9895	1 486
relu	6	Adam	Average	16	0.0287	0.9909	294
relu	6	Adam	Average	32	0.0266	0.9918	469
relu	6	Adam	Average	64	0.0256	0.9928	1 514

Table 4: Complete results from MNIST experimental CNNs (tanh activation).

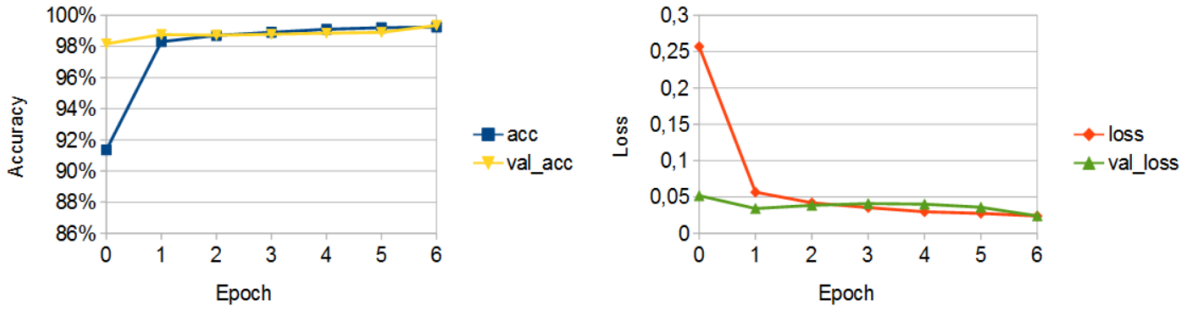
Activ.	**Layers**	**Optimizer**	**Pooling**	**Neurons**	**Loss**	**Accuracy**	**Time (s)**
tanh	2	SGD	Max	16	0.0786	0.9776	222
tanh	2	SGD	Max	32	0.0809	0.9753	342
tanh	2	SGD	Max	64	0.0706	0.9775	811
tanh	2	SGD	Average	16	0.1308	0.9611	363
tanh	2	SGD	Average	32	0.1216	0.9628	501
tanh	2	SGD	Average	64	0.1162	0.9626	888
tanh	2	Adam	Max	16	0.0630	0.9825	335
tanh	2	Adam	Max	32	0.0660	0.9812	541
tanh	2	Adam	Max	64	0.0696	0.9788	1 242
tanh	2	Adam	Average	16	0.0596	0.9826	455
tanh	2	Adam	Average	32	0.0780	0.9765	714
tanh	2	Adam	Average	64	0.0857	0.9734	1 155
tanh	4	SGD	Max	16	0.0487	0.9836	234
tanh	4	SGD	Max	32	0.0441	0.9857	379
tanh	4	SGD	Max	64	0.0363	0.9880	1 102
tanh	4	SGD	Average	16	0.0883	0.9729	410
tanh	4	SGD	Average	32	0.0761	0.9761	599
tanh	4	SGD	Average	64	0.0673	0.9764	1 144
tanh	4	Adam	Max	16	0.0459	0.9870	283
tanh	4	Adam	Max	32	0.0373	0.9871	452
tanh	4	Adam	Max	64	0.0789	0.9725	1 135
tanh	4	Adam	Average	16	0.0509	0.9865	440
tanh	4	Adam	Average	32	0.0524	0.9850	694
tanh	4	Adam	Average	64	0.0530	0.9847	1 150
tanh	6	SGD	Max	16	0.0415	0.9868	246
tanh	6	SGD	Max	32	0.0383	0.9873	474
tanh	6	SGD	Max	64	0.0319	0.9893	1 535
tanh	6	SGD	Average	16	0.0846	0.9736	439
tanh	6	SGD	Average	32	0.0728	0.9777	730
tanh	6	SGD	Average	64	0.0672	0.9795	1 848
tanh	6	Adam	Max	16	0.0435	0.9888	277
tanh	6	Adam	Max	32	0.0335	0.9901	497
tanh	6	Adam	Max	64	0.0672	0.9795	1 679
tanh	6	Adam	Average	16	0.0460	0.9881	312
tanh	6	Adam	Average	32	0.0415	0.9878	591
tanh	6	Adam	Average	64	0.0611	0.9837	1 501

Table 5: Learning process details for the MNIST CNN r2aa16 configuration.

epoch	acc	loss	val_acc	val_loss
0	0.9234	0.2591	0.9693	0.0963
1	0.9655	0.1118	0.9781	0.0711
2	0.9724	0.0872	0.9809	0.0654
3	0.9771	0.0733	0.9821	0.0583
4	0.9792	0.0646	0.9833	0.0552
5	0.9820	0.0563	0.9827	0.0579
6	0.9839	0.0509	0.9857	0.0497

Table 6: Learning process for the MNIST CNN r6am32 configuration.

epoch	acc	loss	val_acc	val_loss
0	0.9136	0.2570	0.9817	0.0517
1	0.9830	0.0565	0.9876	0.0341
2	0.9870	0.0419	0.9872	0.0384
3	0.9890	0.0353	0.9878	0.0409
4	0.9910	0.0296	0.9885	0.0401
5	0.9920	0.0274	0.9890	0.0358
6	0.9925	0.0238	0.9935	0.0238



(a) Accuracy

(b) Loss

Figure 15: Learning process for the MNIST r6am32 configuration.

5.1.4 Analysis and discussion

The results for the experimental CNN models can be considered quite successful. The accuracy ranges from the lowest of **96.11%** (t2sa16) to the highest of **99.36%** (r4aa32). Due to time constraints only a single run through all the configurations was made.

From the gathered data, it would seem the best performing combination for this task would be composed of the **relu activation function, four or six convolutional layers and the adam optimizer function**. These models, but for one exception, scored above 99% accuracy, regardless of the number of neurons in the convolutional layers.

Overall, the results are quite homogeneous and the task might be too simple to reveal differences between the models. The complete set of detailed results is available in the attached CD.

5.2 CIFAR-10

CIFAR-10 is a benchmark problem based on classification on ten specific **object classes**. The dataset consists of 50 000 training images and 10 000 test images.

5.2.1 Data description

The ten classes in CIFAR-10 are **air-plane, auto-mobile, bird, cat, deer, dog, frog, horse, ship, truck**. The images are mutually exclusive. Objects where a classification would be disputable are not included (e.g. a pickup truck). The images are full colour and 32x32 pixels in size [49].

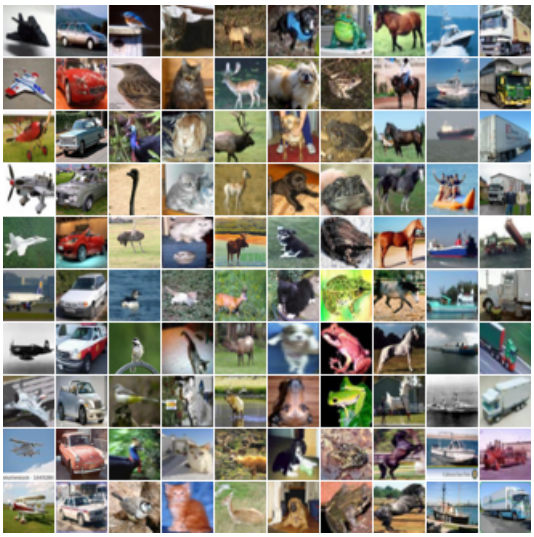


Figure 16: CIFAR-10 database of hand-written digits, a sample.

5.2.2 Methodology

The methodology for this dataset follows the methodology used for the MNIST dataset. This is to test the same model architectures on a much more tasking problem.

Table 7: Complete results from CIFAR-10 experimental CNNs (relu activation).

Activ.	**Layers**	**Optimizer**	**Pooling**	**Neurons**	**Loss**	**Accuracy**	**Time (s)**
relu	2	SGD	Max	16	1.1843	0.5822	229
relu	2	SGD	Max	32	1.0509	0.6258	359
relu	2	SGD	Max	64	1.0629	0.6269	934
relu	2	SGD	Average	16	1.2246	0.5655	367
relu	2	SGD	Average	32	1.2287	0.5666	543
relu	2	SGD	Average	64	1.2022	0.5755	1 001
relu	2	Adam	Max	16	0.9606	0.6911	324
relu	2	Adam	Max	32	1.0774	0.7024	544
relu	2	Adam	Max	64	1.0494	0.6975	1 192
relu	2	Adam	Average	16	0.9521	0.6819	329
relu	2	Adam	Average	32	0.9872	0.6962	603
relu	2	Adam	Average	64	1.0124	0.6756	1 294
relu	4	SGD	Max	16	1.1114	0.6092	219
relu	4	SGD	Max	32	1.1147	0.6066	364
relu	4	SGD	Max	64	1.0274	0.6354	1 132
relu	4	SGD	Average	16	1.2752	0.5381	376
relu	4	SGD	Average	32	1.2154	0.5678	558
relu	4	SGD	Average	64	1.2154	0.5697	1 167
relu	4	Adam	Max	16	0.8313	0.7316	321
relu	4	Adam	Max	32	0.8391	0.7578	563
relu	4	Adam	Max	64	0.8800	0.7412	1 419
relu	4	Adam	Average	16	0.9015	0.7137	271
relu	4	Adam	Average	32	0.8509	0.7422	449
relu	4	Adam	Average	64	0.8626	0.7373	1 230
relu	6	SGD	Max	16	1.2414	0.5664	219
relu	6	SGD	Max	32	1.0962	0.6113	417
relu	6	SGD	Max	64	1.0667	0.6227	1 389
relu	6	SGD	Average	16	1.3665	0.5107	224
relu	6	SGD	Average	32	1.3207	0.5236	574
relu	6	SGD	Average	64	1.2844	0.5391	1 309
relu	6	Adam	Max	16	0.7712	0.7364	272
relu	6	Adam	Max	32	0.7368	0.7508	513
relu	6	Adam	Max	64	0.8987	0.7282	1 498
relu	6	Adam	Average	16	0.8651	0.7047	266
relu	6	Adam	Average	32	0.7747	0.7405	467
relu	6	Adam	Average	64	0.7820	**0.7592**	1 581

Table 8: Complete results from CIFAR-10 experimental CNNs (tanh activation).

Activ.	Layers	Optimizer	Pooling	Neurons	Loss	Accuracy	Time (s)
tanh	2	SGD	Max	16	1.2121	0.5770	230
tanh	2	SGD	Max	32	1.0925	0.6218	404
tanh	2	SGD	Max	64	1.0883	0.6204	765
tanh	2	SGD	Average	16	1.5524	0.4533	232
tanh	2	SGD	Average	32	1.5480	0.4564	368
tanh	2	SGD	Average	64	1.5866	0.4467	865
tanh	2	Adam	Max	16	1.0412	0.6465	349
tanh	2	Adam	Max	32	1.0528	0.6555	623
tanh	2	Adam	Max	64	1.1809	0.5821	1 250
tanh	2	Adam	Average	16	1.5634	0.4525	331
tanh	2	Adam	Average	32	1.3720	0.5116	571
tanh	2	Adam	Average	64	1.6430	0.4049	1 323
tanh	4	SGD	Max	16	1.0039	0.6459	221
tanh	4	SGD	Max	32	0.9736	0.6572	369
tanh	4	SGD	Max	64	0.9935	0.6569	1 149
tanh	4	SGD	Average	16	1.4148	0.4932	222
tanh	4	SGD	Average	32	1.3965	0.5012	373
tanh	4	SGD	Average	64	1.4796	0.4710	1 191
tanh	4	Adam	Max	16	0.9723	0.6746	373
tanh	4	Adam	Max	32	1.0178	0.6784	653
tanh	4	Adam	Max	64	1.0518	0.6332	1 428
tanh	4	Adam	Average	16	1.4756	0.4740	376
tanh	4	Adam	Average	32	1.3719	0.5157	687
tanh	4	Adam	Average	64	1.4990	0.4550	1 328
tanh	6	SGD	Max	16	0.9837	0.6546	225
tanh	6	SGD	Max	32	0.9281	0.6723	542
tanh	6	SGD	Max	64	0.8999	0.6839	1 290
tanh	6	SGD	Average	16	1.3853	0.5014	264
tanh	6	SGD	Average	32	1.3035	0.5297	502
tanh	6	SGD	Average	64	1.3502	0.5166	1 414
tanh	6	Adam	Max	16	0.8710	0.7164	263
tanh	6	Adam	Max	32	0.8631	0.7241	467
tanh	6	Adam	Max	64	1.0886	0.6155	1 483
tanh	6	Adam	Average	16	1.1981	0.5808	270
tanh	6	Adam	Average	32	1.1546	0.5880	456
tanh	6	Adam	Average	64	1.3101	0.5239	1 547

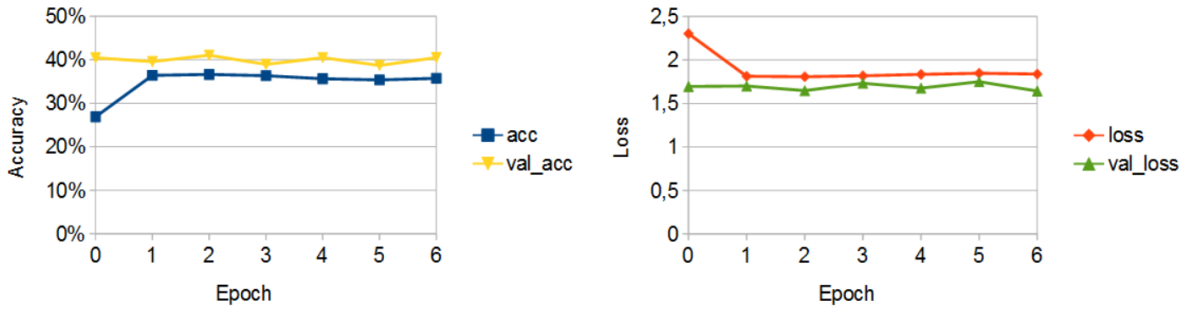
5.2.3 Results

The complete results from all run testes are presented in tables 7 and 8. They show the validation loss, validation accuracy and the time it took to train the model. The below table presents the three main categories for the models' behaviour.

The first example (Tab. 9, Fig. 17), the learning process for a CNN composed of: tanh activation function, two convolutional layers, adam optimizer function, average pooling and sixty-four neurons in the two convolutional layers (encoded as **t2aa64**).

Table 9: Learning process details for the CIFAR-10 CNN t2aa64 configuration.

epoch	acc	loss	val_acc	val_loss
0	0.2687	2.3022	0.4044	1.6950
1	0.3641	1.8124	0.3957	1.7001
2	0.3661	1.8072	0.4104	1.6475
3	0.3633	1.8179	0.3893	1.7318
4	0.3563	1.8346	0.4048	1.6759
5	0.3535	1.8472	0.3869	1.7501
6	0.3573	1.8378	0.4049	1.6430



(a) Accuracy (b) Loss

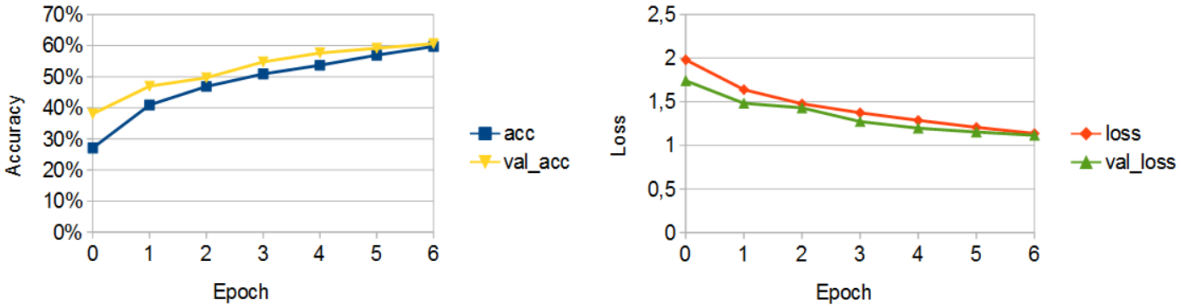
Figure 17: Learning process for the CIFAR-10 CNN t2aa64 configuration.

Secondly (Tab. 10, Fig. 18), a CNN composed of: relu activation function, four convolutional layers, SGD optimizer function, max pooling, thirty-two neurons in the first two and sixty-four neurons in the second two convolutional layers (encoded as **r4sm32**).

The third example (Tab. 11, Fig. 19), a CNN composed of: relu activation function, four convolutional layers, adam optimizer function, max pooling and sixty-four neurons in the first two and one hundred twenty-eight neurons in the second two convolutional layers (encoded as **r4am64**).

Table 10: Learning process details for the CIFAR-10 CNN r4sm32 configuration.

epoch	acc	loss	val_acc	val_loss
0	0.2708	1.9806	0.3807	1.7409
1	0.4092	1.6384	0.4697	1.4820
2	0.4686	1.4760	0.4968	1.4291
3	0.5090	1.3729	0.5479	1.2722
4	0.5370	1.2861	0.5763	1.1963
5	0.5696	1.2059	0.5919	1.1522
6	0.5974	1.1351	0.6066	1.1147



(a) Accuracy

(b) Loss

Figure 18: Learning process for the CIFAR-10 CNN r4sm32 configuration.

5.2.4 Analysis and discussion

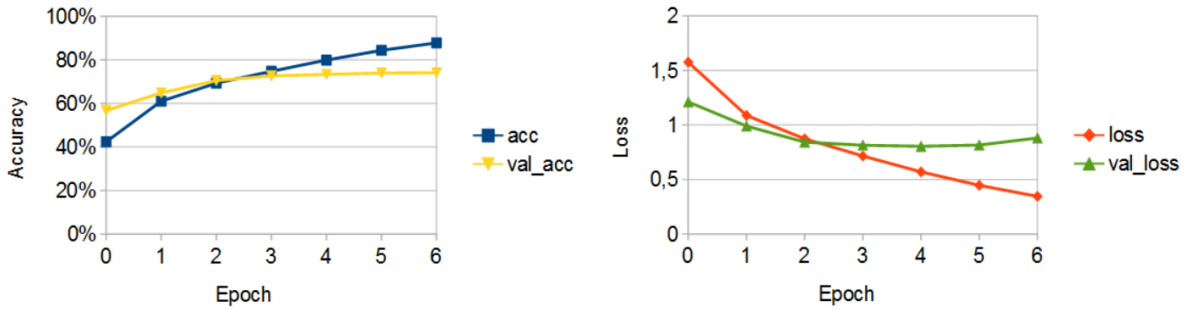
The accuracy for these models range from 40% to 75%. In the three samples presented (t2aa64, r4sm32, r4am64) we can observe the **three categories** in which the models' performances fall into.

In the example **t2aa64**, we can observe the model hitting its performance ceiling right after the second epoch. It then continues to oscillate around the reached accuracy.

The model **r4sm32** reaches its best possible performance around the seventh epoch. We can see all four metrics culminating into joint points. A possible better result could maybe be

Table 11: Learning process details for the CIFAR-10 CNN r4am64 configuration.

epoch	acc	loss	val_acc	val_loss
0	0.4237	1.5752	0.5676	1.2109
1	0.6104	1.0868	0.6486	0.9893
2	0.6930	0.8731	0.7052	0.8424
3	0.7477	0.7148	0.7266	0.8137
4	0.7991	0.5696	0.7333	0.8036
5	0.8438	0.4461	0.7398	0.8153
6	0.8781	0.3449	0.7412	0.8800



(a) Accuracy

(b) Loss

Figure 19: Learning process for the CIFAR-10 CNN r4am64 configuration.

achieved in one or two additional training epochs.

However, the more probable take is the route the example **r4am64** takes. In this example we can see the model reaching its best point by the second epoch. After that, validation accuracy is stagnating and validation loss is slowly increasing. This means the model is over-fitting the training data and is loosing its ability to learn relevant features.

The **best performing models** were composed of: relu activation function, four or six convolutional layers and adam activation function. Both pooling layers and also the 32 and 64 neuron versions performed similarly, overall scoring **above 70%**. Here we can see the same trend as was observed on the MNIST dataset.

6 Experimental results on real biomedical data

For experimental analysis on real biomedical data, the well-known **StarPlus fMRI dataset** was selected. It is based on a study [50], where subjects were shown two symbols (from the selection of a star, a plus or a dollar sign) in a relation to each other (e.g. star above plus) and a sentence describing the picture. The sentence was worded both positively, "*The star is above the plus*", or negatively, "*The star is not below the plus*". The order of the sentence and the picture can alternate. There is always a rest period between the stimulants. The subject was expected to press a button to confirm whether the sentence is *true* or *false*. During the whole process, **fMRI images** of the subject's brain where created with a frequency of one scan every five hundred ms, the test lasted twenty-seven seconds. There is no personal or social information available about the subject, such as age, gender, education level etc. The data was collected by Marcel Just and his colleagues in Carnegie Mellon University's CCBI [51].

Figure 20: Possible example of visual stimulus in the StarPlus study.

This dataset was used as the source for multiple **research papers**. The works by Xuerui Wang, Tom Mitchell et al. [52, 53, 54] are researching the possibilities of decoding the mental state of a single subject and identifying time intervals and brain segments which take part in certain activities. This was later built upon where the recognition capabilities were expanded to **multiple subjects** [55, 56].

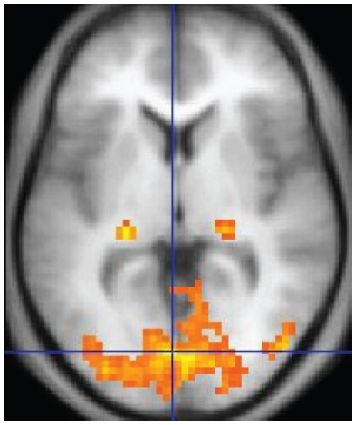


Figure 21: fMRI image example, the highlighted areas mark higher brain activity compared to a control level [57].

These papers were limited due to hard-coded recognition capabilities and necessary expert support. These shortcomings were later addressed under [58]. Further research with modern machine learning methods is still performed on this dataset [59, 60, 61].

6.1 Data description

There are a total of 54 fMRI images, 64x64x8 voxels in size. The data indicates whether the subject was looking at a **sentence**, a **picture** or a **blank** screen during the rest period. The dataset provides many features that can be analysed and the full description is available on the website [62].

The CNN model attempted to learn to be able to **distinguish** between the **mental states** of multiple subjects performing the actions of :

- observing a sentence
- observing a picture
- no visual stimulus

From the available data, only the **first** part (first sixteen images) of the trial was used. The subject was shown a stimulus (eight images), then the stimulus was removed (another eight images). With the second stimulus (after the used images), the brain starts to form a connection with the first stimulus. So the brain response would not not correspond to pure observation activity. This offers a possible extension of this work in the future.

The data available at [51] was broken down and reformatted into a three-dimensional array of a total of **3984 train samples and 800 test samples**. The data is sourced from six test subjects. For validation, the schema "**one-out**" was used, one test subject is singled out and provides the validation data.

6.2 Methodology

To be able to analyse the data, the model had to be modified. While the general **architecture** remains the same as with the MNIST and CIFAR-10 tests, all convolutional and pooling layers were changed from two-dimensional to **three-dimensional**, along with the kernels. The first fully connected layer was changed from 1024 to 128 units due to **memory constraints**. The batch size was reduced to five, also because of memory limitations. The full architecture for a r6sm64 network is shown in table 12.

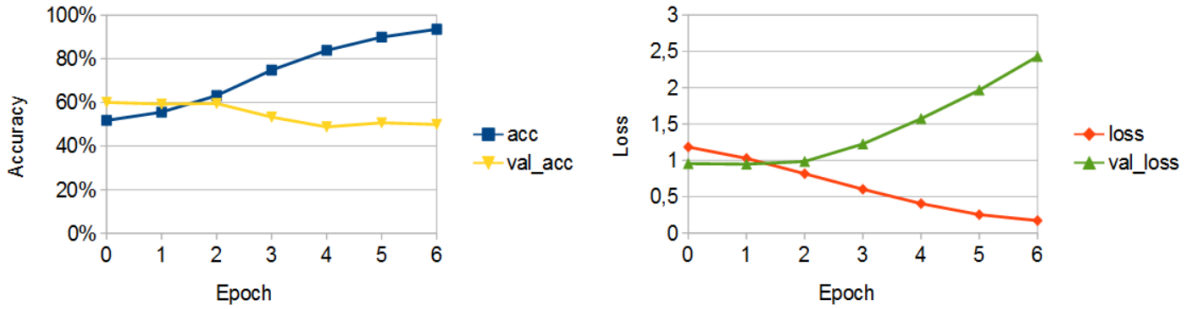
6.3 Results

Table 13 and 14 list the complete results gathered from the fMRI CNN tests. The test results can be broken down into three main categories and an example results is presented from each.

Table 12: Architecture of an r6sm64 CNN model modified for biomedical data.

Layer (type)	Output Shape	Params #
conv3d_1	(None, 64, 64, 8, 64)	1792
conv3d_2	(None, 64, 64, 8, 64)	110656
max_pooling3d_1	(None, 32, 32, 4, 64)	0
conv3d_3	(None, 32, 32, 4, 128)	221312
conv3d_4	(None, 32, 32, 4, 128)	442496
max_pooling3d_2	(None, 16, 16, 2, 128)	0
conv3d_5	(None, 16, 16, 2, 256)	884992
conv3d_6	(None, 16, 16, 2, 256)	1769728
max_pooling3d_3	(None, 8, 8, 1, 256)	0
flatten_1	(None, 16384)	0
dense_1	(None, 128)	2097280
dropout_1	(None, 128)	0
dense_2	(None, 3)	387

The first example(Tab. 15, Fig. 22), a CNN composed of: tanh activation function, two convolutional layers, SGD optimizer function, average pooling and sixty-four neurons in the convolutional layers (encoded as **t2sa64**).



(a) Accuracy (b) Loss

Figure 22: Learning process for the fMRI CNN t2sa64 configuration.

Secondly, shown in table 16 and figure 23, a CNN composed of: tanh activation function, two convolutional layers, SGD optimizer function, max pooling and thirty-two neurons in the convolutional layers (encoded as **t2sm32**).

Table 13: Complete results from fMRI experimental CNNs (ReLu activation)

Activ.	**Layers**	**Optimizer**	**Pooling**	**Neurons**	**Loss**	**Accuracy**	**Time (s)**
relu	2	SGD	Max	16	1.6721	0.5413	661
relu	2	SGD	Max	32	1.7442	0.5613	1 366
relu	2	SGD	Max	64	1.7870	0.5675	3 622
relu	2	SGD	Average	16	1.1903	0.5438	797
relu	2	SGD	Average	32	1.3855	0.5713	1 588
relu	2	SGD	Average	64	1.3215	0.5850	4 362
relu	2	Adam	Max	16	6.4472	0.6000	1 305
relu	2	Adam	Max	32	6.4472	0.6000	1 970
relu	2	Adam	Max	64	6.4472	0.6000	4 764
relu	2	Adam	Average	16	6.4472	0.6000	982
relu	2	Adam	Average	32	1.8270	0.4613	1 858
relu	2	Adam	Average	64	6.4472	0.6000	4 752
relu	4	SGD	Max	16	0.9971	0.5938	1 293
relu	4	SGD	Max	32	1.0469	0.5500	2 024
relu	4	SGD	Max	64	0.9632	0.5888	5 644
relu	4	SGD	Average	16	0.9671	0.5538	958
relu	4	SGD	Average	32	0.9190	0.5738	1 894
relu	4	SGD	Average	64	0.9619	0.5588	5 594
relu	4	Adam	Max	16	6.4472	0.6000	937
relu	4	Adam	Max	32	6.4472	0.6000	2 066
relu	4	Adam	Max	64	0.9508	0.6000	5 089
relu	4	Adam	Average	16	0.9523	0.6000	964
relu	4	Adam	Average	32	0.9506	0.6000	2 050
relu	4	Adam	Average	64	6.4472	0.6000	5 682
relu	6	SGD	Max	16	0.8768	0.6000	951
relu	6	SGD	Max	32	0.8674	0.6025	1 969
relu	6	SGD	Max	64	0.8672	0.6000	6 191
relu	6	SGD	Average	16	0.8650	0.6000	1 375
relu	6	SGD	Average	32	0.8683	0.6000	2 364
relu	6	SGD	Average	64	0.8760	0.6000	6 551
relu	6	Adam	Max	16	0.9507	0.6000	1 231
relu	6	Adam	Max	32	6.4472	0.6000	2 754
relu	6	Adam	Max	64	0.9509	0.6000	6 605
relu	6	Adam	Average	16	0.9504	0.6000	1 027
relu	6	Adam	Average	32	6.4472	0.6000	2 175
relu	6	Adam	Average	64	0.9506	0.6000	6 771

Table 14: Complete results from fMRI experimental CNNs (tanh activation)

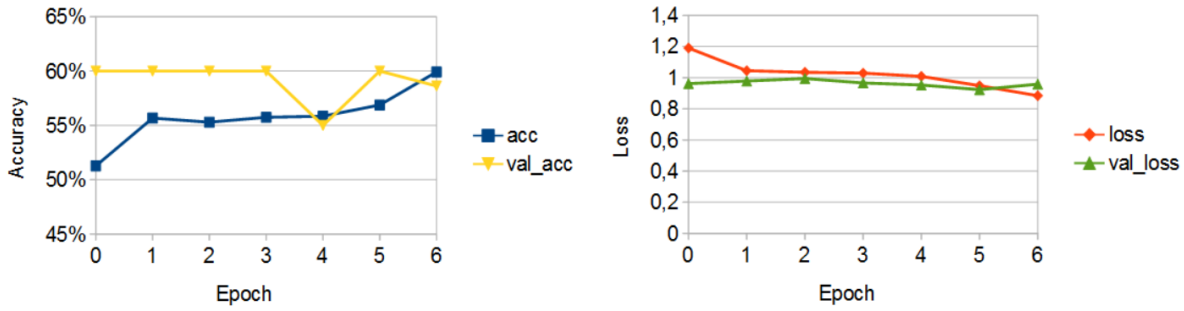
Activ.	**Layers**	**Optimizer**	**Pooling**	**Neurons**	**Loss**	**Accuracy**	**Time (s)**
tanh	2	SGD	Max	32	0.9584	0.5863	1 778
tanh	2	SGD	Max	64	0.9293	0.6000	4 910
tanh	2	SGD	Average	16	2.7388	0.4938	966
tanh	2	SGD	Average	32	2.8475	0.4863	1 479
tanh	2	SGD	Average	64	2.2892	0.4988	4 344
tanh	2	Adam	Max	16	0.9608	0.6000	1 125
tanh	2	Adam	Max	32	0.9529	0.6000	1 833
tanh	2	Adam	Max	64	0.9555	0.6000	5 045
tanh	2	Adam	Average	16	0.9590	0.6000	1 045
tanh	2	Adam	Average	32	0.9528	0.6000	2 006
tanh	2	Adam	Average	64	0.9609	0.6000	5 423
tanh	4	SGD	Max	16	1.6219	0.5025	1 037
tanh	4	SGD	Max	32	2.3403	0.3988	1 832
tanh	4	SGD	Max	64	2.5732	0.3775	4 707
tanh	4	SGD	Average	16	1.2098	0.5388	1 243
tanh	4	SGD	Average	32	1.3426	0.5738	2 040
tanh	4	SGD	Average	64	1.2652	0.5750	5 605
tanh	4	Adam	Max	16	0.9846	0.6000	1 249
tanh	4	Adam	Max	32	0.9628	0.6000	2 109
tanh	4	Adam	Max	64	0.9911	0.6000	5 130
tanh	4	Adam	Average	16	0.9551	0.6000	1 315
tanh	4	Adam	Average	32	0.9647	0.6000	2 118
tanh	4	Adam	Average	64	0.9510	0.6000	5 665
tanh	6	SGD	Max	16	1.2378	0.4525	1 410
tanh	6	SGD	Max	32	1.3521	0.4900	2 283
tanh	6	SGD	Max	64	1.6480	0.4638	6 487
tanh	6	SGD	Average	16	0.9619	0.5750	1 340
tanh	6	SGD	Average	32	0.9363	0.5950	2 337
tanh	6	SGD	Average	64	0.9596	0.5838	6 510
tanh	6	Adam	Max	16	0.9544	0.6000	1 404
tanh	6	Adam	Max	32	0.9542	0.6000	2 326
tanh	6	Adam	Max	64	0.9625	0.6000	6 878
tanh	6	Adam	Average	16	0.9532	0.6000	1 394
tanh	6	Adam	Average	32	0.9627	0.6000	2 327
tanh	6	Adam	Average	64	0.9619	0.6000	6 882

Table 15: Learning process for the fMRI CNN t2sa64 configuration.

epoch	acc	loss	val_acc	val_loss
0	0.5173	1.1833	0.6000	0.9545
1	0.5552	1.0270	0.5925	0.9475
2	0.6313	0.8161	0.5950	0.9843
3	0.7482	0.6021	0.5325	1.2249
4	0.8386	0.4045	0.4875	1.5730
5	0.8996	0.2524	0.5063	1.9657
6	0.9355	0.1711	0.4988	2.4289

Table 16: Learning process for the fMRI CNN t2sm32 configuration.

epoch	acc	loss	val_acc	val_loss
0	0.5128	1.1907	0.6000	0.9621
1	0.5567	1.0449	0.6000	0.9787
2	0.5530	1.0347	0.6000	0.9947
3	0.5575	1.0293	0.6000	0.9663
4	0.5585	1.0085	0.5500	0.9530
5	0.5688	0.9487	0.6000	0.9235
6	0.5991	0.8843	0.5863	0.9584



(a) Accuracy

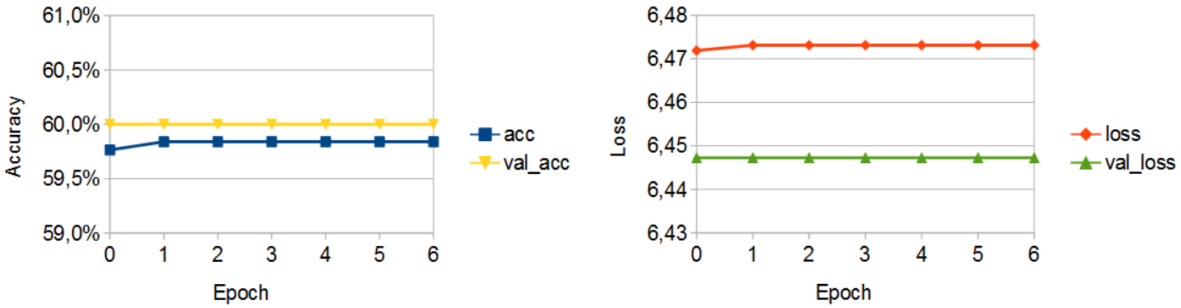
(b) Loss

Figure 23: Learning process for the fMRI CNN t2sm32 configuration.

And the third example, table 17, figure 24, a CNN composed of: relu activation function, two convolutional layers, adam optimizer function, max pooling and sixteen neurons in the convolutional layers (encoded as **r2am16**).

Table 17: Learning process for the fMRI CNN r2am16 configuration.

epoch	acc	loss	val_acc	val_loss
0	0.5976	6.4719	0.6000	6.4472
1	0.5984	6.4731	0.6000	6.4472
2	0.5984	6.4731	0.6000	6.4472
3	0.5984	6.4731	0.6000	6.4472
4	0.5984	6.4731	0.6000	6.4472
5	0.5984	6.4731	0.6000	6.4472
6	0.5984	6.4731	0.6000	6.4472



(a) Accuracy

(b) Loss

Figure 24: Learning process for the fMRI CNN r2am16 configuration.

As a special comparison the tables 18 and 19 show the progress of the models **r6sm32** and **t2sm16**.

Table 18: Learning process for the fMRI CNN r6sm32 configuration.

epoch	acc	loss	val_acc	val_loss
0	0.5946	0.9765	0.6000	0.9694
1	0.5984	0.9613	0.6000	0.9517
2	0.5984	0.9588	0.6000	0.9525
3	0.5984	0.9527	0.6000	0.9345
4	0.5984	0.9320	0.6000	0.9380
5	0.5969	0.9147	0.6000	0.8623
6	0.5996	0.8956	0.6025	0.8674

6.4 Analysis and discussion

From the first example, the test case **t2sa64** shows clear signs of **over-fitting**, training accuracy increasing above 90%, while validation accuracy dropping beyond 50%. A similar behaviour can be observer for the model t4sm16.

Table 19: Learning process for the fMRI CNN t2sm16 configuration.

epoch	acc	loss	val_acc	val_loss
0	0.5183	1.1638	0.6000	1.0501
1	0.5650	1.0179	0.6000	0.9284
2	0.5858	0.9458	0.6188	0.8854
3	0.6142	0.8751	0.6025	0.9454
4	0.6403	0.8040	0.5850	0.9617
5	0.6810	0.7400	0.5963	1.0831
6	0.7033	0.6875	0.6025	1.1559

The second category, detailed in the second example, are models who oscillate around very similar values, but never stepping out of their boundaries. The **t2sm32** model's test and validation accuracy remains around 60% during the whole training process. The model **r4sa64** follows this trend with small signs of over-fitting during the last epochs.

The **majority of tests** (the third sample) show static results during the whole training process, the model is unable to learn general features that would be applicable outside the training cases.

The possible reason for this behaviour might be the limitations of the constructed architecture, there is not enough breadth and depth to extract and learn classification features. The reduction of fully-connected layer size that was done due to hardware limitations is also a contributing factor.

The lack of change in the loss values also shows that the approximation functions are stuck and unable to escape the local function area. Intervention through the increase of the learning rate, while probably decreasing the overall accuracy between the models, might provide some more accurate models.

Lastly, the "one-out" schema used for training and test data selection can be the possible reason behind such static results, assuming the test data differs from the training data in a way the makes effective learning of test data features more difficult.

The two best performing networks, with 60.25% accuracy, were **r6sm32** and **t2sm16**. Their learning progress is shown in tables 18 and 19. As visible in the data, the model r6sm32 follows the third type of networks trend with only a small improvement in the last epoch. The model t2sm16 shows more variance from the second epoch onwards. Overall, it is clear that the models need to further extended and fine-tuned to be successful in this task.

7 Conclusions and future work

The thesis provided an introduction to the state of art of machine learning and convolutional neural networks. The proposed CNN architectures were successfully tested against benchmark problems and real-world fMRI data. The models' performance was satisfactory on the benchmarks, but the analysis of more complex fMRI data revealed issues with the design of the networks and the testing approach.

However, certain models were found that can be utilized and build upon in possible future tests. These will need to be expanded and fine tuned to achieve a higher classification accuracy. It was also found that a consumer grade notebook graphical card imposes severe limits on the size of a CNN model and a realistic scope of testing.

Future work that can be inferred from the results would entail: deepening of CNN models, implementation of early stopping to save on wasteful computing time, broadening the scope of testing for additional hyper-parameters, evaluation of alternative testing hardware, e.g. GPU servers or cloud computing.

Overall, based on the gathered data, it can be concluded that CNNs are a capable tool for image patter recognition.

References

[1] NEWSWIRE, MultiVu-PR, [no date]. *Survey: 80 Percent of Enterprises Investing in AI, But Cite Significant Challenges Ahead*. Multivu [online]. [Accessed 2018-03-10]. Available from: https://www.multivu.com/players/English/8075951-teradata-state-of-artificial-intelligence-ai-for-enterprises/

[2] FAGGELLA, D., 2016. *Machine Learning Healthcare Applications - 2018 and Beyond*. TechEmergence [online]. 29 August 2016. [Accessed 2018-04-02]. Available from: https://www.techemergence.com/machine-learning-healthcare-applications/

[3] SCHMIDHUBER, J., 2015. *Deep Learning in Neural Networks: An Overview*. Neural Networks [online]. 61, 85–117. ISSN 08936080. Available from: doi:10.1016/j.neunet.2014.09.003.

[4] ROSENBLATT, F., 1958. *The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain*. Psychological Review. 65–386.

[5] LECUN, Y, J. S. DENKER, R. E. HOWARD, W. HUBBARD and L. D. JACKEL, 1989. *Backpropagation Applied to Handwritten Zip Code Recognition*. Neural Computation. 1(4), 541–551.

[6] O'SHEA, K. and R. NASH, 2015. *An Introduction to Convolutional Neural Networks*. arXiv:1511.08458 [cs] [online]. [Accessed 2018-04-24]. Available from: http://arxiv.org/abs/1511.08458

[7] RAMASWAMY, S. and S. MALL, [no date]. *Pulmonary Nodule Classification with Convolutional Neural Networks*. 9.

[8] SHIN, H., H. R. ROTH, M. GAO, L. LU, Z. XU, I. NOGUES, J. YAO, D. MOLLURA and R. M. SUMMERS, 2016. *Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning*. arXiv:1602.03409 [cs] [online]. [Accessed 2018-04-26]. Available from: http://arxiv.org/abs/1602.03409

[9] PEREIRA, S., A. PINTO, V. ALVES and C. A. SILVA, 2016. *Brain Tumor Segmentation Using Convolutional Neural Networks in MRI Images*. IEEE Transactions on Medical Imaging [online]. 35(5), 1240–1251. ISSN 0278-0062. Available from: doi:10.1109/TMI.2016.2538465

[10] SINGLETON, M. J., 2009. *Functional Magnetic Resonance Imaging*. The Yale Journal of Biology and Medicine. 82(4), 233. ISSN 0044-0086.

[11] Watson Oncology, [no date]. *Memorial Sloan Kettering Cancer Center* [online]. [Accessed 2018-03-11]. Available from: https://www.mskcc.org/about/innovative-collaborations/watson-oncology

[12] SIWICKI, B., 2017. *Future-proofing AI: Embrace machine learning now because healthcare adoption is picking up speed*, 2017. Healthcare IT News [online]. [Accessed 2018-04-02]. Available from: http://www.healthcareitnews.com/news/future-proofing-ai-embrace-machine-learning-now-because-healthcare-adoption-picking-speed

[13] BALLARD, D. H. and C. M. BROWN, 1982. *Computer Vision*. 1st ed. B.m.: Prentice Hall Professional Technical Reference. ISBN 0-13-165316-4.

[14] PUBMEDDEV, [no date]. *Home - PubMed - NCBI* [online] [Accessed 2018-04-23]. Available from: https://www.ncbi.nlm.nih.gov/pubmed/

[15] HASTIE, T., J. FRIEDMAN and R. TIBSHIRANI, 2001. *The Elements of Statistical Learning* [online]. New York, NY: Springer New York. Springer Series in Statistics [Accessed 2018-03-10]. ISBN 978-1-4899-0519-2. Available from: doi:10.1007/978-0-387-21606-5

[16] GOODFELLOW, I., Y. BENGIO and A. COURVILLE, 2016. *Deep Learning*. B.m.: MIT Press.

[17] REDMAN, T. C., 2018. *If Your Data Is Bad, Your Machine Learning Tools Are Useless*. Harvard Business Review [online] [Accessed 2018-04-18]. Available from: https://hbr.org/2018/04/if-your-data-is-bad-your-machine-learning-tools-are-useless

[18] BENGIO, Y., 2009. *Learning Deep Architectures for AI*. Foundations and Trends in Machine Learning [online]. 2(1), 1–127. ISSN 1935-8237. Available from: doi:10.1561/2200000006

[19] LECUN, Y., G. HINTON and Y. BENGIO, 2015. *Deep Learning*. Nature. 521, 436–444.

[20] BISHOP, C. M., 2006. *Pattern recognition and machine learning*. New York: Springer. Information science and statistics. ISBN 978-0-387-31073-2.

[21] GEMAN, S., E. BIENENSTOCK and R. DOURSAT, 1992. *Neural Networks and the Bias/Variance Dilemma. Neural Computation* [online]. 4(1), 1–58. Available from: doi:10.1162/neco.1992.4.1.1

[22] LANGE, N., C. M. BISHOP and B. D. RIPLEY, 1997. *Neural Networks for Pattern Recognition. Journal of the American Statistical Association* [online]. 92(440), 1642. ISSN 01621459. Available from: doi:10.2307/2965437

[23] SHALEV-SHWARTZ, S. and S. BEN-DAVID, 2014. *Understanding Machine Learning: From Theory to Algorithms* [online]. Cambridge: Cambridge University Press [Accessed 2018-03-10]. ISBN 978-1-107-29801-9. Available from: doi:10.1017/CBO9781107298019

[24] OPENCLIPART-VECTORS, [no date]. *Free vector graphic: Brain, Neuron, Nerves, Cell - Free Image on Pixabay - 2022398* [online]. [Accessed 2018-03-18]. Available from: https://pixabay.com/en/brain-neuron-nerves-cell-science-2022398/

[25] KRENKER, A., J. BEŠTER and A. KOS, 2011. *Introduction to the Artificial Neural Networks. Artificial Neural Networks - Methodological Advances and Biomedical Applications* [online]. [Accessed 2018-04-01]. Available from: doi:10.5772/15751

[26] BENGIO, Y, 2009. *Learning Deep Architectures for AI. Foundations* [online]. 2, 1–55. Available from: doi:10.1561/2200000006

[27] RUDER, S., 2016. *An overview of gradient descent optimization algorithms.* arXiv:1609.04747 [cs] [online]. [Accessed 2018-04-01]. Available from: http://arxiv.org/abs/1609.04747

[28] ZERODAMAGE, Gradient_descent png: The original uploader was Olegalexandrov at English Wikipediaderivative work:, 2012. *An illustration of the gradient descent method. I graphed this with Matlab* [online]. 7 August 2012. [Accessed 2018-04-02]. Available from: https://commons.wikimedia.org/wiki/File:Gradient_descent.svg

[29] KINGMA, D. P. and J. BA, 2014. *Adam: A Method for Stochastic Optimization.* arXiv:1412.6980 [cs] [online]. [Accessed 2018-04-02]. Available from: http://arxiv.org/abs/1412.6980

[30] LITJENS, G., T. KOOI, B. E. BEJNORDI, A. A. A. SETIO, F. CIOMPI, M. GHAFOORIAN, J. A. W. M. VAN DER LAAK, B. VAN GINNEKEN and C. I. SÁNCHEZ, 2017. *A survey on deep learning in medical image analysis.* Medical Image Analysis [online]. 42, 60–88. ISSN 13618415. Available from: doi:10.1016/j.media.2017.07.005

[31] DE BRUIJNE, M., 2016. *Machine learning approaches in medical image analysis: From detection to diagnosis.* Medical Image Analysis [online]. 33, 20th anniversary of the Medical Image Analysis journal (MedIA), 94–97. ISSN 1361-8415. Available from: doi:10.1016/j.media.2016.06.032

[32] KNIGHT, W., [no date]. *There's a big problem with AI: even its creators can't explain how it works.* MIT Technology Review [online]. [Accessed 2018-04-03]. Available from: https://www.technologyreview.com/s/604087/the-dark-secret-at-the-heart-of-ai/

[33] FUKUSHIMA, K., 1980. *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.* Biological Cybernetics [online]. 36(4), 193–202. ISSN 0340-1200, 1432-0770. Available from: doi:10.1007/BF00344251

[34] PLOTKE, M., 2013. *File:Vd-Orig.png - Wikipedia* [online] [Accessed 2018-04-13]. Available from: https://en.wikipedia.org/wiki/File:Vd-Orig.png

[35] PLOTKE, M., 2013. *File:Vd-Edge3.png - Wikipedia* [online] [Accessed 2018-04-13]. Available from: https://en.wikipedia.org/wiki/File:Vd-Edge3.png

[36] GÉRON, A., 2018. *Neural networks and deep learning* [online]. B.m.: O'Reilly Media, Inc. [Accessed 2018-04-17]. ISBN 9781692037347. Available from: http://www.safaribooksonline.com/library/view/neural-networks-and/9781492037354/

[37] SPRINGENBERG, J. T., A. DOSOVITSKIY, T. BROX and M. RIEDMILLER, 2014. *Striving for Simplicity: The All Convolutional Net*. arXiv:1412.6806 [cs] [online]. [Accessed 2018-04-13]. Available from: http://arxiv.org/abs/1412.6806

[38] LI, F., J. JOHNSON and S. YEUNG, [no date]. *CS231n Convolutional Neural Networks for Visual Recognition*. [online]. [Accessed 2018-03-27]. Available from: http://cs231n.github.io/convolutional-networks/

[39] LECUN, Y., L. BOTTOU, Y. BENGIO and P. HAFFNER, 1998. *Gradient-Based Learning Applied to Document Recognition*. Proceedings of the IEEE. 86(11), 2278–2324.

[40] KRIZHEVSKY, A., I. SUTSKEVER and G. E. HINTON, 2012. *ImageNet Classification with Deep Convolutional Neural Networks*. In: F. PEREIRA, C. J. C. BURGES, L. BOTTOU a K. Q. WEINBERGER, ed. Advances in Neural Information Processing Systems 25 [online]. B.m.: Curran Associates, Inc., s. 1097–1105 [Accessed 2018-04-13]. Available from: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[41] SZEGEDY, C., W. LIU, Y. JIA, P. SERMANET, S. REED, D. ANGUELOV, D. ERHAN, V. VANHOUCKE and A. RABINOVICH, 2014. *Going Deeper with Convolutions*. arXiv:1409.4842 [cs] [online]. [Accessed 2018-04-13]. Available from: http://arxiv.org/abs/1409.4842

[42] SIMONYAN, K. and A. ZISSERMAN, 2014. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv:1409.1556 [cs] [online]. [Accessed 2018-04-14]. Available from: http://arxiv.org/abs/1409.1556

[43] HE, K., X. ZHANG, S. REN and J. SUN, 2015. *Deep Residual Learning for Image Recognition*. arXiv:1512.03385 [cs] [online]. [Accessed 2018-04-14]. Available from: http://arxiv.org/abs/1512.03385

[44] ORR, G. and K. MÜLLER, ed., 1998. *Neural networks: tricks of the trade*. Berlin; New York: Springer. Lecture notes in computer science, 1524. ISBN 978-3-540-65311-0.

[45] CAO, X., [no date]. *A practical theory for designing very deep convolutional neural networks* [online]. Available from: https://kaggle2.blob.core.windows.net/forum-message-attachments/69182/2287/A%20practical%20theory%20for%20designing%20very%20deep%20convolutional%20neural%20networks.pdf

[46] SMITH, L. N. and N. TOPIN, 2016. *Deep Convolutional Neural Network Design Patterns*. arXiv:1611.00847 [cs] [online]. [Accessed 2018-04-13]. Available from: http://arxiv.org/abs/1611.00847

[47] LECUN, Y., C. CORTES and C. BURGES, [no date]. *MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges*. [online]. [Accessed 2018-03-27]. Available from: http://yann.lecun.com/exdb/mnist/

[48] HINTON, G. E., N. SRIVASTAVA, A. KRIZHEVSKY, I. SUTSKEVER and R. R. SALAKHUTDINOV, 2012. *Improving neural networks by preventing co-adaptation of feature detectors*. arXiv:1207.0580 [cs] [online]. [Accessed 2018-03-10]. Available from: http://arxiv.org/abs/1207.0580

[49] KRIZHEVSKY, A., [no date]. *Learning Multiple Layers of Features from Tiny Images*. [online]. [Accessed 2018-04-05]. Available from: http://www.cs.toronto.edu/ kriz/learning-features-2009-TR.pdf

[50] REICHLE, E. D., P. A. CARPENTER and M. A. JUST, 2000. *The Neural Bases of Strategy and Skill in Sentence–Picture Verification*. Cognitive Psychology [online]. 40(4), 261–295. ISSN 00100285. Available from: doi:10.1006/cogp.2000.0733

[51] *StarPlus fMRI data*, [no date]. [online]. [Accessed 2018-04-07]. Available from: http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-81/www/

[52] WANG, X. and T. M. MITCHELL, 2002. *Detecting Cognitive States Using Machine Learning*. 11.

[53] MITCHELL, T. M., R. HUTCHINSON, M. A. JUST, R. S. NICULESCU, F. PEREIRA and X. WANG, 2003. *Classifying Instantaneous Cognitive States from fMRI Data*. American Medical Informatics Association Annual Symposium. 5.

[54] MITCHELL, T. M., R. HUTCHINSON, R. S. NICULESCU, F. PEREIRA, X. WANG, M. A. JUST and S. NEWMAN, 2004. *Learning to Decode Cognitive States from Brain Images*. Machine Learning [online]. 57(1/2), 145–175. ISSN 0885-6125. Available from: doi:10.1023/B:MACH.0000035475.85309.1b

[55] WANG, X., T. M. MITCHELL and R. HUTCHINSON, 2003. *Using Machine Learning to Detect Cognitive States across Multiple Subjects* [online]. Available from: https://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-73/www/papers/xuerui_kdd.pdf

[56] WANG, X., R. HUTCHINSON and T. M. MITCHELL, [no date]. *Training fMRI Classifiers to Detect Cognitive States across Multiple Human Subjects*. 8.

[57] OPENSTAX, 2016. *Version 8.25 from the Textbook* [online]. 18. May 2016. [Accessed 2018-04-26]. Available from: https://commons.wikimedia.org/wiki/File:1206_FMRI.jpg

[58] RAMISH, J., 2004. *Learning Common Features from fMRI Data of Multiple Subjects*. 11.

[59] RAMASANGU, H. and N. SINHA, 2016. *Cognitive state classification using transformed fMRI data*. arXiv:1604.05413 [cs] [online]. [Accessed2018-04-07]. Available from: http://arxiv.org/abs/1604.05413

[60] PANDEY, P., B. K. JHA and N. SINHA, 2016. *Analyzing Cognitive States Using fMRI Data*. Procedia Computer Science [online]. 90, 35–41. ISSN 18770509. Available from: doi:10.1016/j.procs.2016.07.007

[61] FAN, M. and C. CHOU, 2016. *Exploring stability-based voxel selection methods in MVPA using cognitive neuroimaging data: a comprehensive study*. Brain Informatics [online]. 3(3), 193–203. ISSN 2198-4018. Available from: doi:10.1007/s40708-016-0048-0

[62] MITCHELL, T. M., 2005. *Data file and data structure documentation for the fMRI starplus study*. [online]. March 2005. [Accessed 2018-04-11]. Available from: http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-81/www/README-data-documentation.txt

A Attachments

Attachment n. 1: CD with all result and source files

B Attached CD's folder structure

The CD attached to this thesis contains the accompanying files organized in to following fashion:

Path	Description
/Application	Software and Documentation
/Result data	Results gathered from test runs
/Thesis	Thesis